Pivoting a Company Towards GitOps and Continuous Delivery

A Technical Practices Coach's Tale for the Misbegotten

DEVOPS
PRO
EUROPE

Howard Deiner

Who Am I?

On Requirements Gathering

What About Planning?

Development environments

Development Infrastructure

On Coding Practices

Let's Talk About Testing

Production Infrastructure

Developing a Learning Organization

The Tale for the Misbegotten

WHO AM I?

About Me

Programming since high school in 1976 (some days, I'd figure it out). (Look after him 1978)

Working full-time since 1979 (while still a junior in college)

Have had the pleasure of working at all levels from a single contributor to executive), both commercial and scientific.

Have been an Agile Coach since 2003 (so few of us back then that it seemed that we all knew one another)

To be successful back then, you had to be both a process and extreme programming Coach.

I still feel the same way that I did back in 2003.
- Day now, we have so many companies that will tell you that it makes you "agile".
- We're sort of become what we fought against.
- Seems like the biggest differences are the words we use for our same ideas and processes.
- Things got sort of hard for me at meetups 2013, where the ideas just slipped out.

What I Believe In

The Agile Fluency Model (at least, that's what I coach towards).

I believe in real pragmatism, and understanding how systems thinking can be applied to organizations.

I believe that it usually doesn't matter if your organization is even mildly inefficient, so long as they are effective in empathizing with, and then delighting their customers.

I'm a strict believer that using Lean Thinking, used against organizational issues, produces the best results. At least, some call so scientific method, and the idea of a feedback cycle, powered by hypothesis and observation, will help us make informed decisions about how to approach organizations and their ineffectivenesses.

Why Do I Do What I Do?

I wonder about this all the time. Especially when I get frustrated. Which, sadly, is not an uncommon state.

I do what I do primarily because I enjoy helping others learn and appreciate knowledge.

I'm not particularly interested in power. Because power only works in an environment where people respect the office (remember, I'm a child of the 60s and I still believe that one should challenge authority!). Rather, I enjoy things when people initially enjoy my presence and thoughts and we can have interesting interactions. When we both learn from one another, that works for me.

I also do what I do because the alternative is to retire. And I've pledged that once I retire, I want to study law again (story of my JD from 1984), pass the bar, and work for free for the American Civil Liberties Union as an Immigration Lawyer. Keeping active in this community means that I don't have to retire, and therefore don't have to do that crazy amount of hard work studying for the bar.

And What's My Take on DevOps?

Developers and operations people working more closely together? No. Necessary, but not sufficient.

Use of a bunch of DevOps tools? No. Again, use of tools is necessary, but not sufficient. ("Man is above the apes because he has learned to extend his grip.")

Some special letters after your name from taking a course? No. (I usually can tell if I share a belief system with someone who wants to LinkIn with me by the length of the letters after their name.)

Nope. I believe that DevOps is all about establishing a DevOps culture in the organization. It's a return to a pragmatic way of delighting the customers we empathize with. It's very much an Agile 2.0.

The DevOps Mindset

No special, unusual, and risky "go lives". Sustainable engineering means no more work induced sleep depravation!

Work that we do during Development goes into our customers hands as a normal part of the Development cycle.

We will use good engineering practices.
We will use some rather specialized DevOps tools.
We will collaborate closely with those who monitor sites that our customers use.

But the essence is that we work in a completely full-stack non-silo fashion. (Which is sort of how I used to work in the late 70s on Cobol Mainframes versus what I see now.)

The DevOps Culture

Increased collaboration between development and operations, brought about because of the recognition of a shared responsibility to create and delight the organization's customers.

Autonomous and self-reliant teams to allow work to progress without the Lean Wastes related to coordination, excessive planning, waiting, defects, inventory, motion, etc.

Building quality into the process, rather than trying test it into the product by "shifting testing left" as far as possible.

Establishing feedback loops between customers and developers, developers and operations, operations and audit/regulatory, etc. to help promote Continuous Improvement in a learning Organization.

Understanding and practicing the solid quality based iterative development of everything, as contrasted with extensive requirements/analysis/design/code/test project plans that are commonly used, regardless of any Agile sounding words and tools used to express them.

Specifically, DevOps is not about buying a lot of expensive "Enterprise DevOps" tools based on vendor promises of how wonderful/magical things will be with their expensive tools and then trying to make all the tools work together somehow.

About Me

Programming since High School in 1968 (some day, I'll figure it out) [cool Retro IBM 1130]

Working full-time since 1975 (while still a junior in college)

Have had the pleasure of working at all levels (individual contributor to executive), both commercial and scientific

Have been an Agile Coach since 2003 (so few of us back then that it seemed that we all knew one another)

To be successful back then, you had to be both a process and eXtreme Programming Coach

I still feel the same way that I did back in 2003
- But now, we have so many companies that will sell you stuff to make you "Agile"
- We've sort of become what we fought against
- Seems like the biggest differences are the words we use for our old style silos and processes
- Things got out of hand for me at AgileIndy 2017, where the words just slipped out

Howard Deiner

Version 1.0

Copyright @1954-????
All rights reserved.

About Me

Programming since High School in 1968 (some day, I'll figure it out) [cool Retro IBM 1130]

Working full-time since 1975 (while still a junior in college)

Have had the pleasure of working at all levels (individual contributor to executive), both commercial and scientific

Have been an Agile Coach since 2003 (so few of us back then that it seemed that we all knew one another)

To be successful back then, you had to be both a process and eXtreme Programming Coach

I still feel the same way that I did back in 2003
- But now, we have so many companies that will sell you stuff to make you "Agile"
- We've sort of become what we fought against
- Seems like the biggest differences are the words we use for our old style silos and processes
- Things got out of hand for me at AgileIndy 2017, where the words just slipped out

Howard Deiner
Version 1.0

Copyright @1954-????
All rights reserved.

What I Believe In

The Agile Fluency Model (a least, that's what I coach towards)

I believe in real Pragmatism, and understanding how systems thinking can be applied to organizations.

I believe that it usually doesn't matter if your organization is even mildly inefficient, so long as they are effective in empathizing with, and then delighting their customers.

I'm a strict believer that using Lean Thinking, used against organizational issues, produces the best results. At least, some call to scientific method, and the idea of a feedback cycle, powered by hypothesis and observation, will help us make informed decisions about how to approach organizations and their ineffectivenesses.

THE AGILE FLUENCY MODEL
CHART YOUR AGILE PATHWAY

"Scaling agile" always sounds to me like "scaling small-batch, hand-crafted artisanal beer." You end up with Bud Light

Andy Hunt
Pragmatic programmer

THE AGILE FLUENCY™ MODEL

CHART YOUR AGILE PATHWAY

Positive
Inclusive
Promotes Improvement
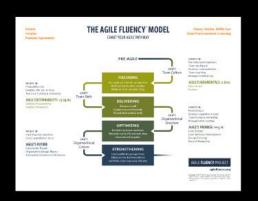
Fluency: Routine, Skillful Ease
Comes From Investment In Learning

PRE-AGILE

SHIFT
Team Culture

INVEST IN:
Full-time team members
Team workspace
Business representation
Team coaching
Management training

AGILE FUNDAMENTALS: 2-6 MO.
Basic Scrum
Kanban

FOCUSING
See progress from biz perspective
Redirect teams when needed
Work on most valuable thing

INVEST IN:
Productivity dip
DevOps, UX, etc. in team
Technical training & mentoring

AGILE SUSTAINABILITY: +3-24 MO.
Extreme Programming
DevOps Movement

SHIFT
Team Skills

DELIVERING
Release at will
Capture value frequently
Reveal obstructions early

SHIFT
Organizational Structure

INVEST IN:
Market focus
Business expertise in team
Team business ownership
Management coaching

AGILE'S PROMISE: +1-5 YR.
Lean Startup
Lean Software Development
Design Thinking
Beyond Budgeting

OPTIMIZING
Excellent product decisions
Eliminate hand-offs and wait time
Innovation/disruption

INVEST IN:
Inventing new practices
Cross-organization focus

AGILE'S FUTURE
Complexity Theory
Organization Design Theory
Alternative Governance Structures

SHIFT
Organizational Culture

STRENGTHENING
Cross-pollinate perspectives
Stimulate market innovations
Optimize cross-org value streams

AGILE FLUENCY PROJECT

agilefluency.org

" "Scaling agile" always sounds to me like "scaling small-batch, hand-crafted artisanal beer." You end up with Bud Light

Andy Hunt
Pragmatic programmer

What I Believe In

"Scaling agile" always sounds to me like "scaling small-batch, hand-crafted artisanal beer." You end up with Bud Light

Andy Hunt
Pragmatic programmer

The Agile Fluency Model (a least, that's what I coach towards)

I believe in real Pragmatism, and understanding how systems thinking can be applied to organizations.

I believe that it usually doesn't matter if your organization is even mildly inefficient, so long as they are effective in empathizing with, and then delighting their customers.

I'm a strict believer that using Lean Thinking, used against organizational issues, produces the best results. At least, some call to scientific method, and the idea of a feedback cycle, powered by hypothesis and observation, will help us make informed decisions about how to approach organizations and their ineffectivenesses.

Why Do I Do What I Do?

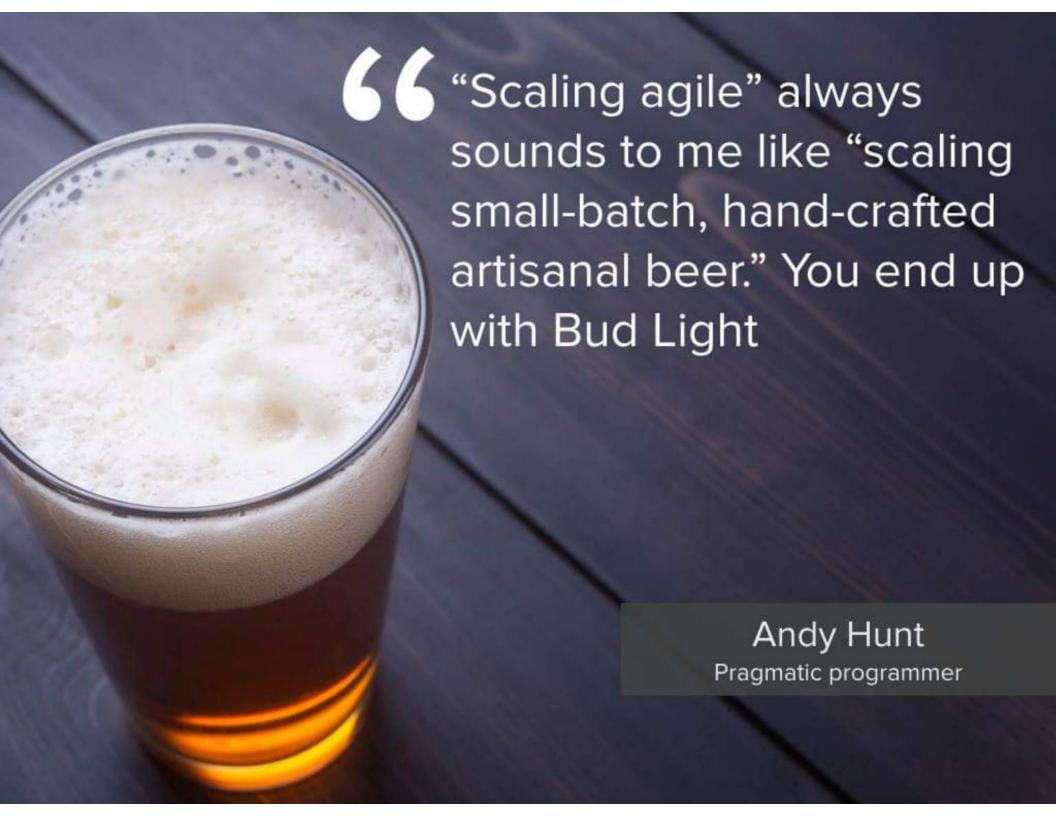I wonder about this all the time. Especially when I get frustrated. Which, sadly, is not an uncommon state.

I do what I do primarily because I enjoy helping others learn and appreciate knowledge.

I'm not particularly interested in power. Because power only works in an environment where people respect the office (remember. I'm a child of the 60s and I still believe that one should challenge authority!). Rather, I enjoy things when people mutually enjoy my presence and thoughts and we can have interesting interactions. When we both learn from one another, that works for me.

I also do what I do because the alternative is to retire. And I've pledged that once I retire, I want to study law again (Story of my JD from 1984), pass the bar, and work for free for the American Civil Liberties Union as an Immigration Lawyer. Keeping active in this community means that I don't have to retire, and therefore don't have to do that crazy amount of hard work studying for the bar.

And What's My Take on DevOps?

Developers and operations people working more closely together? No. Necessary, but not sufficient.

Use of a bunch of DevOps tools? No. Again, use of tools is necessary, but not sufficient. ("Man is above the apes because he has learned to extend his grip.")

Some special letters after your name from taking a course? No. (I usually can tell if I share a belief system with someone who wants to LinkIn with me by the length of the letters after their name.)

Nope. I believe that DevOps is all about establishing a DevOps culture in the organization. It's a return to a pragmatic way of delighting the customers we empathize with. It's very much an Agile 2.0.

And What's My Take on DevOps?

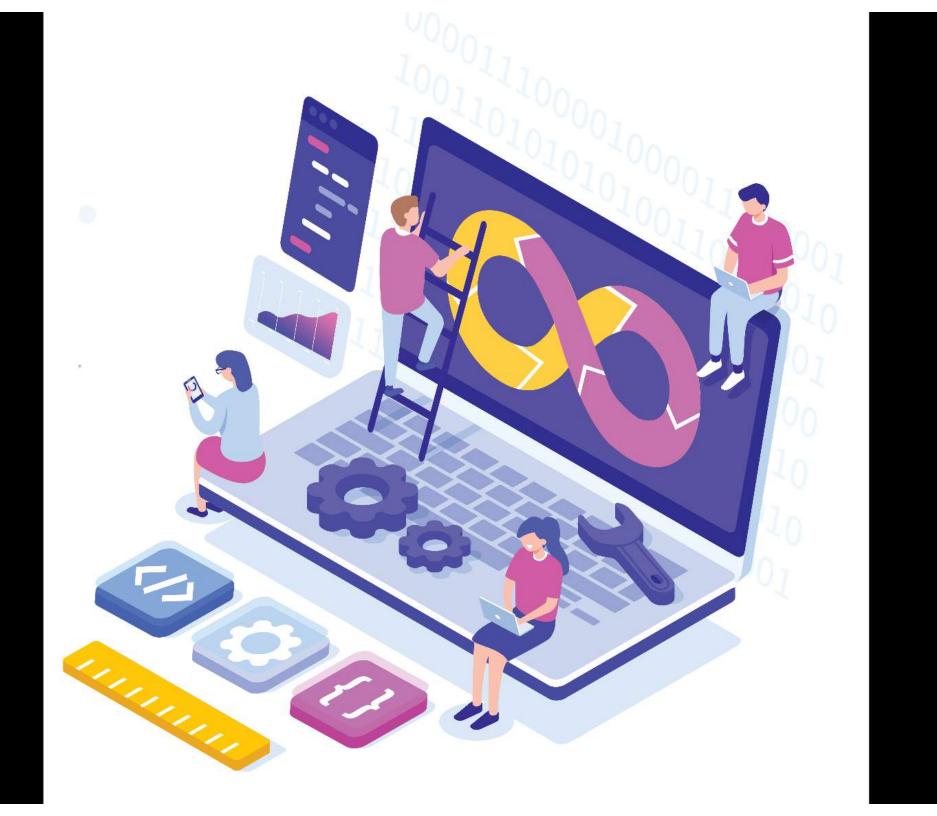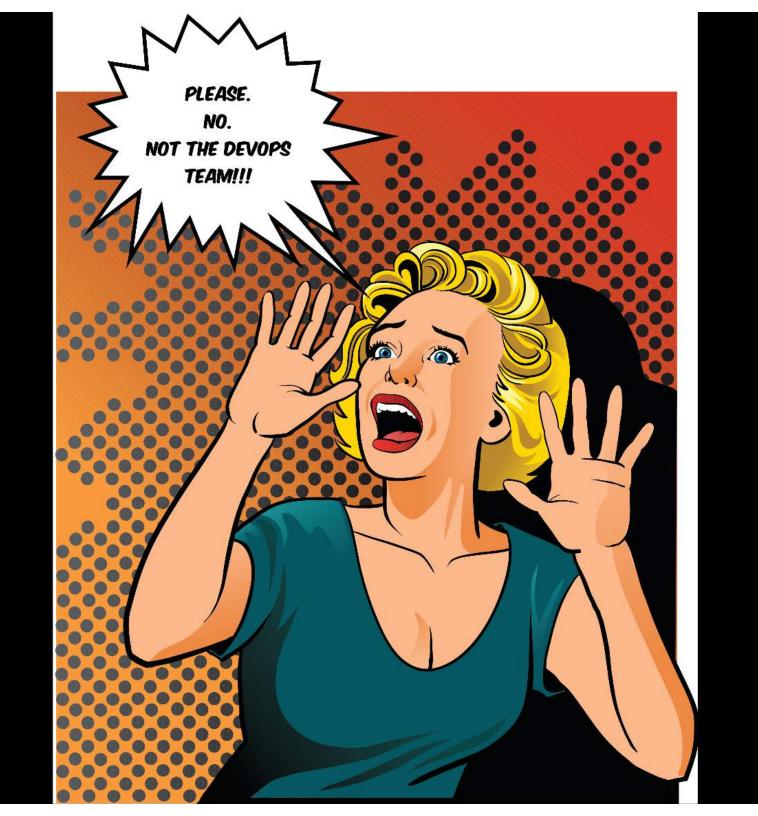Developers and operations people working more closely together? No. Necessary, but not sufficient.

Use of a bunch of DevOps tools? No. Again, use of tools is necessary, but not sufficient. ("Man is above the apes because he has learned to extend his grip.")

Some special letters after your name from taking a course? No. (I usually can tell if I share a belief system with someone who wants to LinkIn with me by the length of the letters after their name.)

Nope. I believe that DevOps is all about establishing a DevOps culture in the organization. It's a return to a pragmatic way of delighting the customers we empathize with. It's very much an Agile 2.0.

The DevOps Mindset

No special, unusual, and risky "go lives". Sustainable engineering means no more work induced sleep depravation!

Work that we do during Development goes into our customers hands as a normal part of the Development cycle.

We will use good engineering practices.
We will use some rather specialized DevOps tools.
We will collaborate closely with those who monitor sites that our customers use.

But the essence is that we work in a completely full-stack non-silo fashion. (Which is sort of how I used to work in the late 70s on Cobol Mainframes versus what I see now.)

The DevOps Culture

Increased collaboration between development and operations, brought about because of the recognition of a shared responsibility to excite and delight the organization's customers.

Autonomous and self-reliant teams to allow work to progress without the Lean Wastes related to coordination, excessive planning, waiting, defects, inventory, motion, etc.

Building quality into the process, rather than trying test it into the product by "shifting testing left" as far as possible.

Establishing feedback loops between customers and developers, developers and operations, operations and audit/regulatory, etc. to help promote Continuous Improvement in a Learning Organization.

Understanding and practicing the solid quality based iterative development of everything, as contrasted with extensive requirements/analysis/design/code/test project plans that are commonly used, regardless of any Agile sounding words and tools used to express them.

Specifically, DevOps is not about buying a lot of expensive "Enterprise DevOps" tools based on vendor promises of how wonderful/magical things will be with their expensive tool and then trying to make all the tools work together somehow.

WHO AM I?

About Me

Programming since high school in 1978 (some say, the figure is off). Took Algol from 1978.

Working full-time since 1985 (while still a junior in college).

Have had the pleasure of working at all levels (from a very low functionality to executives), both commercial and scientific.

Have been an Agile Coach since 2003 (so few of us back then that it seems that we all knew one another).

To be successful back then, you had to be both a Process and eXtreme Programming Coach.

I still feel the same way that I did back in 2003.
- Day back, we have so many companies that will tell you that it makes you "Agile"
- We're tired of hearing what we taught against
- Seems like the biggest differences were the words we use for our same ideas and processes
- Things got kind of hard for me at meetings 2012, where the words just slipped out

What I Believe In

The Agile Fluency Model (at least, that's what I coach towards).

I believe in real pragmatism, and understanding how systems thinking can be applied to organizations.

I believe that it usually doesn't matter if your organization is even mildly inefficient, so long as they are effective in empathizing with, and then delighting their customers.

I'm a strict believer that using Lean Thinking, used against organizational issues, produces the best results. At least, some call so scientific method, and the idea of a feedback cycle, powered by hypothesis and observation, will help us make informed decisions about how to approach organizations and their ineffectivenesses.

Why Do I Do What I Do?

I wonder about this all the time. Especially when I get frustrated. Which, sadly, is not an uncommon state.

I do what I do primarily because I enjoy helping others learn and appreciate knowledge.

I'm not particularly interested in power. Because power only works in an environment where people respect the office (remember, I'm a child of the 60s and I still believe that one should challenge authority!). Rather, I enjoy things when people initially enjoy my presence and thoughts and we can have interesting interactions. When we both learn from one another, that works for me.

I also do what I do because the alternative is to retire. And I've pledged that once I retire, I want to study law again (story of my JD from 1984), pass the bar, and work for free for the American Civil Liberties Union as an Immigration Lawyer. Keeping active in this community means that I don't have to retire, and therefore don't have to do that crazy amount of hard work studying for the bar.

And What's My Take on DevOps?

Developers and operations people working more closely together? No. Necessary, but not sufficient.

Use of a bunch of DevOps tools? No. Again, use of tools is necessary, but not sufficient. ("Man is above the apes because he has learned to extend his grip.")

Some special letters after your name from taking a course? No. (I usually can tell if I share a belief system with someone who wants to LinkIn with me by the length of the letters after their name.)

Nope. I believe that DevOps is all about establishing a DevOps culture in the organization. It's a return to a pragmatic way of delighting the customers we empathize with. It's very much an Agile 2.0.

The DevOps Mindset

No special, unusual, and risky "60 lives". Sustainable engineering means no more work induced sleep deprivation!

Work that we do during development goes into our customers hands as a normal part of the development cycle.

We will use good engineering practices.
We will use some rather specialized DevOps tools.
We will collaborate closely with those who monitor sites that our customers use.

But the essence is that we work in a completely full-stack non-silo fashion. (Which is sort of how I used to work in the late 70s on Cobol Mainframes versus what I see now.)

The DevOps Culture

Increased collaboration between development and operations, brought about because of the recognition of a shared responsibility to create and delight the organization's customers.

Autonomous and self-reliant teams to allow work to progress without the Lean Wastes related to coordination, excessive planning, waiting, defects, inventory, motion, etc.

Building quality into the process, rather than trying test it into the product by "shifting testing left" as far as possible.

Establishing feedback loops between customers and developers, developers and operations, operations and audit/regulatory, etc. to help promote Continuous Improvement in a learning organization.

Understanding and practicing the solid quality based iterative development of everything, as contrasted with extensive requirements/analysis/design/code/test project plans that are commonly used, regardless of any Agile sounding words and tools used to express them.

Specifically, DevOps is not about buying a lot of expensive "Enterprise DevOps" tools based on vendor promises of how wonderful/magical things will be with their expensive tools and them trying to make all the tools work together somehow.

Who Am I?

On Requirements Gathering

What About Planning?

Development environments

Development Infrastructure

On Coding Practices

Let's Talk About Testing

Production Infrastructure

Developing a Learning Organization

The Tale for the Misbegotten

Questions,
Answers,
Debate,
Disputes

On Requirements Gathering

We're Agile. We Don't Need no stinking Requirements!

(The Treasure of the Sierra Madre, Warner Brothers, 1948)

Steve McConnell, in Code Complete, states: "The first prerequisite you need to fulfill before beginning construction is a clear statement of the problem the system is supposed to solve."

But we're Agile! Do we need to gather requirements?

- Surely, probably so.
- Frankly, we should constantly collaborate with our customers that we know what they need.
- But, it's usually not that simple. Especially when you're not the only developer who works for their customer.

We actually have two problems:

- Why doing what the user wants.
- Doing what the user wants.
- What?

The answer is at the intersection of "Curse of Knowledge" from the user's standpoint versus doing only what the user asks for. (Hard it Better Challenge and Ford's "quick" or "If I asked people what they wanted, it was faster horses." (Innovation occurs with true empathy of the customer's needs).

Doing Requirements Right

The good news is that we can ensure that we create what we said we were going to create with an eye towards scientific method and a virtuous feedback loop to test the hypothesis that we created what we said we did by automated behavior driven development testing, using things like Gherkin to allow both customers and developers communicate that shared vision of what we are working from.

As a side benefit, this gives us an automated regression suite for free, to ensure continued software quality.

Well, that's not how many people see requirements gathering. Even my customer falls back on this time after time.

Why? I put the blame squarely on people "Reverting to Form" when stressed, and performing Commitment Driven Development. (Side story of Chicago incorporation for this).

Why it takes practice to get past the way that we were taught that software has to be built...

The ancient formula of "Analyze, Design, Develop, and Test" (Waterfall) is now cloaked in modern agile sounding words. "Product Increment", "Sprint Objective", "Story."

We still yearn to look for complete solutions to needs way before we really and deeply understand the real needs.

An Agile approach would be to do the simplest thing possible in the direction of where we think a solution lays. Have the customer test that to reinforce our hypothesis of what a solution looks like. And iteratively (not incrementally) put forward a solution.

In Commitment Driven Development, we ask people to estimate, hopefully with high accuracy, the cost for something something, in the initiative's hope that we can optimize the effort by scaling across the organizational silos so everyone is doing their little part in making the solution happen.

But by doing so, we miss the big picture by forgetting that we don't really know what the solution is. We probably don't even know what the problem really is?

We're Agile. We Don't Need no stinking Requirements!

[The Treasure of the Sierra Madre, Warner Brothers, 1948]

Steve McConnell, in Code Complete, states: "The first prerequisite you need to fulfill before beginning construction is a clear statement of the problem the system is supposed to solve."

But we're Agile! Do we need to gather requirements?
- Sadly, probably so.
- Ideally, we should empathize so much with our customers that we know what they need.
- But, it's usually not that simple. Especially when you're not the only developer who works for their customer.

We actually have two worries.
- Not doing what the user wants.
- Doing what the user wants.
- What?

The answer is at the intersection of Curse of Knowledge from the user's standpoint versus doing only what the user asks for. [Peanut Butter Challenge and Ford's "quote" of "If I asked people what they wanted, it was faster horses." (Innovation occurs with true empathy of the customer's needs)]

We're Agile. We Don't Need no stinking Requirements!

[The Treasure of the Sierra Madre, Warner Brothers, 1948]

Steve McConnell, in Code Complete, states: "The first prerequisite you need to fulfill before beginning construction is a clear statement of the problem the system is supposed to solve."

But we're Agile! Do we need to gather requirements?
- Sadly, probably so.
- Ideally, we should empathize so much with our customers that we know what they need.
- But, it's usually not that simple. Especially when you're not the only developer who works for their customer.

We actually have two worries.
- Not doing what the user wants.
- Doing what the user wants.
- What?

The answer is at the intersection of Curse of Knowledge from the user's standpoint versus doing only what the user asks for. [Peanut Butter Challenge and Ford's "quote" of "If I asked people what they wanted, it was faster horses." (Innovation occurs with true empathy of the customer's needs)]

We're Agile. We Don't Need no stinking Requirements!

[The Treasure of the Sierra Madre, Warner Brothers, 1948]

Steve McConnell, in Code Complete, states: "The first prerequisite you need to fulfill before beginning construction is a clear statement of the problem the system is supposed to solve."

But we're Agile! Do we need to gather requirements?
- Sadly, probably so.
- Ideally, we should empathize so much with our customers that we know what they need.
- But, it's usually not that simple. Especially when you're not the only developer who works for their customer.

We actually have two worries.
- Not doing what the user wants.
- Doing what the user wants.
- What?

The answer is at the intersection of Curse of Knowledge from the user's standpoint versus doing only what the user asks for. [Peanut Butter Challenge and Ford's "quote" of "If I asked people what they wanted, it was faster horses." (Innovation occurs with true empathy of the customer's needs)]

Doing Requirements Right

The good news is that we can ensure that we create what we said we were going to create with an eye towards scientific method and a virtuous feedback loop to test the hypothesis that we created what we said we did by automated behavior driven development testing, using things like Gherkin to allow both customers and developers communicate that shared vision of what we are working from.

As a side benefit, this gives us an automated regression suite for free, to ensure continued software quality

Well, that's not how many people see requirements gathering. Even my customer falls back on this time after time.

Why? I put the blame squarely on people "Reverting to Form" when stressed, and performing Commitment Driven Development. [Side story of Chicago inspiration for this]
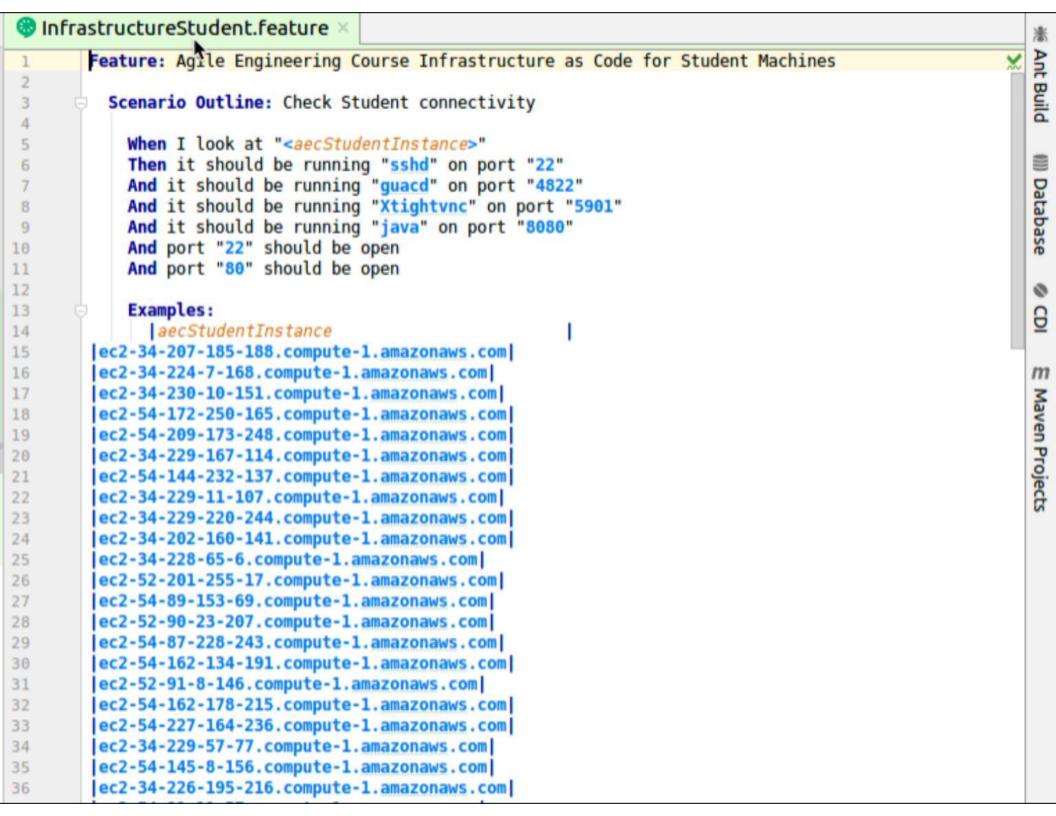
```gherkin
Feature: Density Altitude

  On June 19 and 20, 2017, many flights were cancelled at Arizona's Phoenix Sky Harbor airport
  because it was "too hot."  Really?  The engineering reason has to do with the fact that the amount
  of lift generated by a wing is related to the air that it flies in.  When the air becomes "thin"
  because of heat and humidity (water vapor is less dense than air: that's why clouds tend to float
  above us), they produce less lift, and require higher speeds through the air to produce the lift
  needed to allow the airplanes to take off.  Higher speeds require longer runways, and at a certain
  point, the heat is just too high to have enough safety on any particular runway to allow a safe depature.

  With Global Warming becoming all too real, pliots need to keep their eye on the density altitude (a measure
  of what the heat and humidity would translate into for "standard" atmospheric conditions) as they evaluate
  how much runway is needed for a safe takeoff and landing.  If you're interested in the details of how
  density altitude is calculated, take a look at https://en.wikipedia.org/wiki/Density_altitude.

  There are many online resources that pilots can use to calculate density altitude.  We'd like to test them!

  Scenario Outline: Density altitude checking
    Given I'm using the following browsers
      | firefox         |
      | chrome          |
    And I'm using the following appplications
      | Pilot Friend          |
      | Shelquist Engineering |
    When the elevation is "<elevation>" "<elevationUnits>"
    And the temperature is "<temperature>" "<temperatureUnits>"
    And the altimeter is "<altimeter>" "<altimeterUnits>"
    And the dew point is "<dewPoint>" "<dewPointUnits>"
    Then the density altitude should be "<densityAltitude>" "<densityAltitudeUnits>"

    Examples:
```

elevation	elevationUnits	temperature	temperatureUnits	altimeter	altimeterUnits	dewPoint	dewPoi
182	feet	80	F	30.02	in Hg	70	F
182	feet	60	F	29.95	in Hg	40	F
30	meters	25	C	1000	hPa	20	C

```gherkin
Feature: Agile Engineering Course Infrastructure as Code for Student Machines

  Scenario Outline: Check Student connectivity

    When I look at "<aecStudentInstance>"
    Then it should be running "sshd" on port "22"
    And it should be running "guacd" on port "4822"
    And it should be running "Xtightvnc" on port "5901"
    And it should be running "java" on port "8080"
    And port "22" should be open
    And port "80" should be open

    Examples:
      |aecStudentInstance                        |
|ec2-34-207-185-188.compute-1.amazonaws.com|
|ec2-34-224-7-168.compute-1.amazonaws.com|
|ec2-34-230-10-151.compute-1.amazonaws.com|
|ec2-54-172-250-165.compute-1.amazonaws.com|
|ec2-54-209-173-248.compute-1.amazonaws.com|
|ec2-34-229-167-114.compute-1.amazonaws.com|
|ec2-54-144-232-137.compute-1.amazonaws.com|
|ec2-34-229-11-107.compute-1.amazonaws.com|
|ec2-34-229-220-244.compute-1.amazonaws.com|
|ec2-34-202-160-141.compute-1.amazonaws.com|
|ec2-34-228-65-6.compute-1.amazonaws.com|
|ec2-52-201-255-17.compute-1.amazonaws.com|
|ec2-54-89-153-69.compute-1.amazonaws.com|
|ec2-52-90-23-207.compute-1.amazonaws.com|
|ec2-54-87-228-243.compute-1.amazonaws.com|
|ec2-54-162-134-191.compute-1.amazonaws.com|
|ec2-52-91-8-146.compute-1.amazonaws.com|
|ec2-54-162-178-215.compute-1.amazonaws.com|
|ec2-54-227-164-236.compute-1.amazonaws.com|
|ec2-34-229-57-77.compute-1.amazonaws.com|
|ec2-54-145-8-156.compute-1.amazonaws.com|
|ec2-34-226-195-216.compute-1.amazonaws.com|
```

Doing Requirements Right

The good news is that we can ensure that we create what we said we were going to create with an eye towards scientific method and a virtuous feedback loop to test the hypothesis that we created what we said we did by automated behavior driven development testing, using things like Gherkin to allow both customers and developers communicate that shared vision of what we are working from.

As a side benefit, this gives us an automated regression suite for free, to ensure continued software quality

Well, that's not how many people see requirements gathering. Even my customer falls back on this time after time.

Why? I put the blame squarely on people "Reverting to Form" when stressed, and performing Commitment Driven Development. [Side story of Chicago inspiration for this]

Why it takes practice to get past the way that we were taught that software has to be built...

The ancient formula of "Analyze, Design, Develop, and Test" (Waterfall) is now cloaked in modern Agile sounding words. "Product Increment". "Sprint Objective". "Story."

We still yearn to look for complete solutions to needs way before we really and deeply understand the real needs.

An Agile approach would be to do the simplest thing possible in the direction of where we think a solution lays. Have the customer test that to reinforce our hypothesis of what a solution looks like. And iteratively (not incrementally) put forward a solution.

In Commitment Driven Development, we ask people to estimate, hopefully with high accuracy, the cost for something something, in the misguided hope that we can optimize the effort by scaling across the organizational silos so everyone is doing their little part in making the solution happen.

But by doing so, we miss the big picture by forgetting that we don't really know what the solution is. We probably don't even know what the problem really is!

On Requirements Gathering

We're Agile. We Don't Need no stinking Requirements!

(The Treasure of the Sierra Madre, Warner Brothers, 1948)

Steve McConnell, in Code Complete, states: "The first prerequisite you need to fulfill before beginning construction is a clear statement of the problem the system is supposed to solve."

But we're Agile! Do we need to gather requirements?
- Sadly, probably so.
- Ideally, we should constantly collaborate with our customers that we know what they need.
- But, it's usually not that simple. Especially when you're not the only developer who works for their customer.

We certainly have two problems:
- Why doing what the user wants.
- Doing what the user wants.
- What?

The answer is at the intersection of "Curse of knowledge" from the user's standpoint versus doing only what the user asks for. (read it Busted Challenge and find it) "laugh" or "If I asked people what they wanted, it was faster horses." (innovation occurs with true empathy of the customer's needs).

Doing Requirements Right

The good news is that we can ensure that we create what we said we were going to create with an eye towards scientific method and a virtuous feedback loop to test the hypothesis that we created what we said we did by automated behavior driven development testing, using things like Gherkin to allow both customers and developers communicate that shared vision of what we are working from.

As a side benefit, this gives us an automated regression suite for free, to ensure continuous software quality.

Well, that's not how many people see requirements gathering. Even my customer falls back on this time after time.

Why? I put the blame squarely on people "Reverting to Form" when stressed, and performing Commitment Driven Development. (Side story of Chicago inspiration for this).

Why it takes practice to get past the way that we were taught that software has to be built...

The ancient formula of "Analyze, Design, Develop, and Test" (Waterfall) is now cloaked in modern agile sounding words. "Product Increment", "Sprint Objective", "Story."

We still yearn to look for complete solutions to needs way before we really and deeply understand the real needs.

An Agile approach would be to do the simplest thing possible in the direction of where we think a solution lays. Have the customer test that to reinforce our hypothesis of what a solution looks like. And iteratively (not incrementally) put forward a solution.

In Commitment Driven Development, we ask people to estimate, hopefully with good accuracy, the cost for something, in the initiative's hope that we can optimize the effort by scaling across the organizational silos so everyone is doing their little part in making the solution happen.

But by doing so, we miss the big picture by forgetting that we don't really know what the solution is. We probably don't even know what the problem really is!

Who Am I?

On Requirements Gathering

What About Planning?

Development environments

Development Infrastructure

On Coding Practices

Let's Talk About Testing

Production Infrastructure

Developing a Learning Organization

The Tale for the Misbegotten

Questions,
Answers,
Debate,
Deserve

What About Planning?

Why Do Organizations Like to Plan So Much?
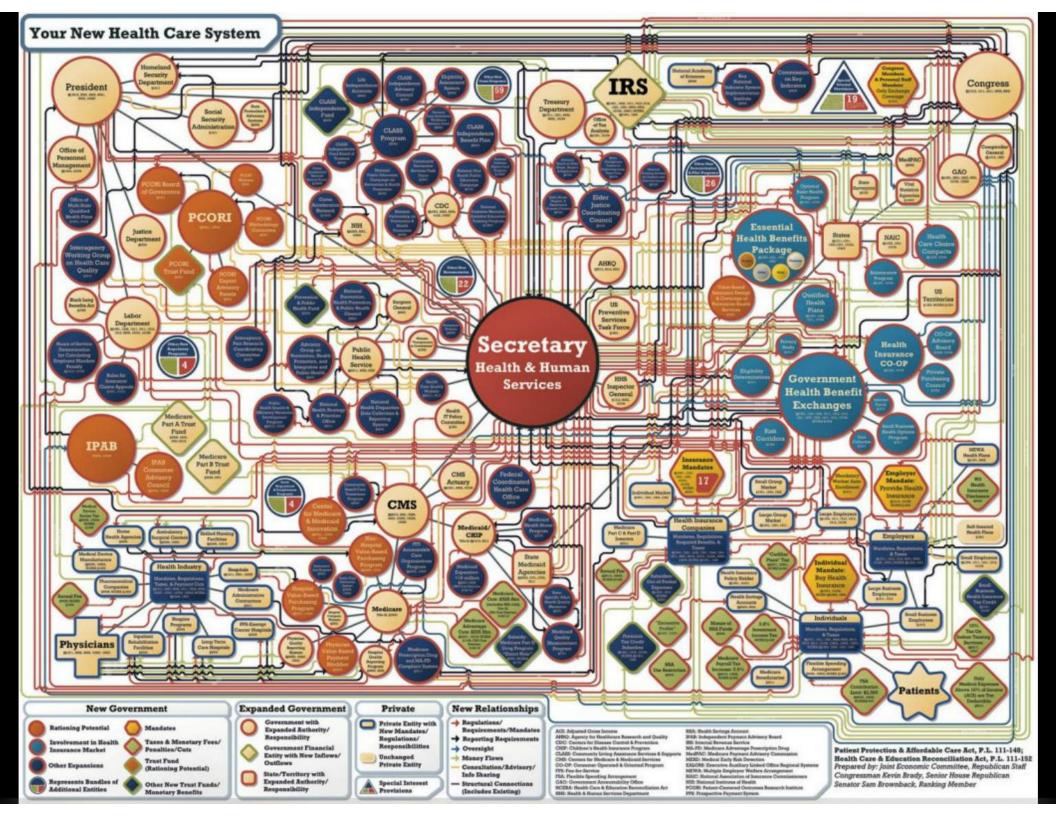

I really wish I knew.

Organizations that I've dealt with have a deep history of planning. They know the waterfall and sort of like it.
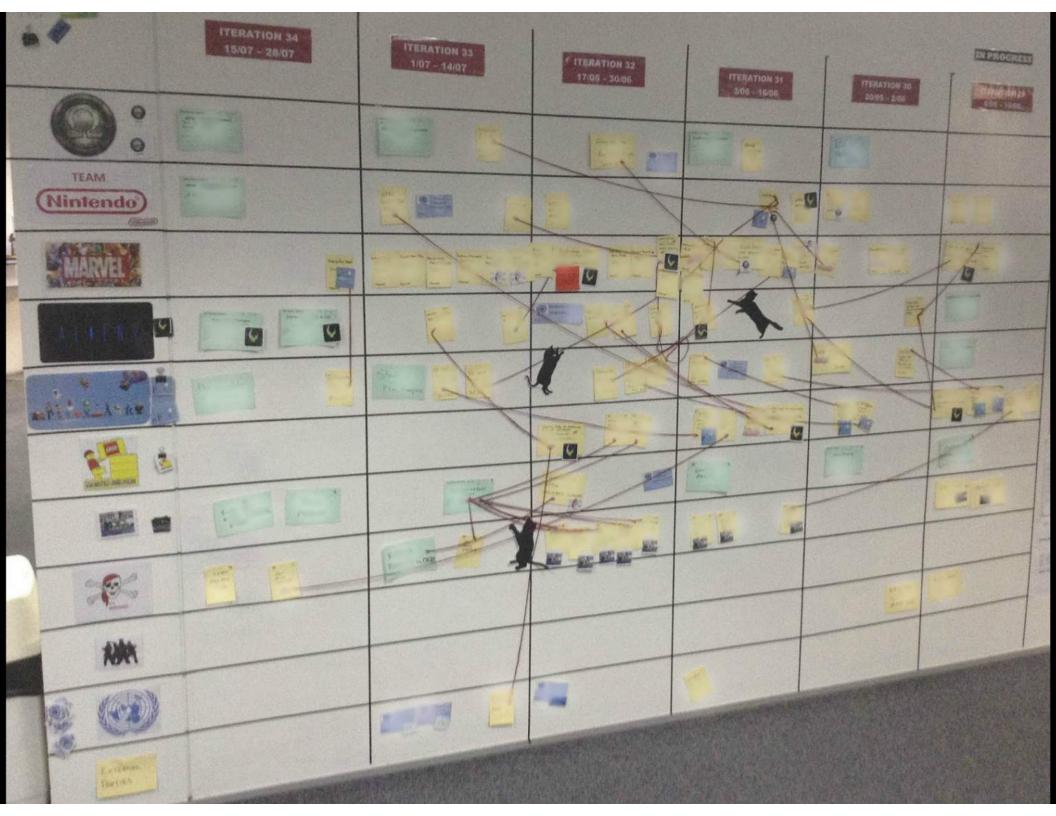
I feel that ultimately, the need to plan is rooted with organizational distrust in the developers.

And there's that need to globally act efficiently that organizations always seem to have. They feel that the seeds they put in place give people the chance to experiment today, and become the experts at it, making them a more efficient on the future.

But then they become the only ones qualified to do the work. Which leads to the vicious cycle of more planning and more Lean Wastes.

Planning for Fixing Time - the Hardening Sprints

Usually, developers in organizations I start to coach are put at a great distance from their ultimate customers. They have "Product Owners", "Inner Project Owners", and a plethora of other words that are supposed to mean that these people understand the products.

Most of the organizations don't even use "Product" people to staff that role. They use "IT" or "Development" staff. IMHO, these are usually "Project Managers" with another name.

As a coach, I find myself constantly correcting teams and organizations on this topic. They constantly tell me that "we can't release a story that doesn't have all of these features, because there is no value to our customers without everything being prepared".

They forget that the value is in avoiding Lean Wastes for Inventory, Waiting, Over Production, Over Processing, and Defects, all these lists of things constantly occur with the going plans for getting a project done.

In fact, I find that teams are perfectly content on having 6 sprint releases (12 weeks), with the final, 2 sprints reserved for "hardening." That's a 33% penalty to allow for the defects and bugs that occur due to late integration breakage. And some organizations add on another sprint for planning a release.

That's just unacceptable waste.

Lean Wastes and the Meeting Culture

And speaking of Lean Wastes, I still have an enormous problem with my mind with a meeting culture.

I have a client who is not unlike other similar businesses. They love to meet. And if they can't meet in person, they solve their wheels.

I work closely with the leadership of all development of the organization. A guy who has over 150 people who report to him himself. I like to kick off meetings calendars like a birthday maze out of multi-colored Post-It stones. It's mostly meetings where people act important, or progress, and collusing on "Post-Its".

There are several issues and present dangers in to this:
- Group decisions are usually mediocre at best.
- There is keeping others programmatic decisions unless that manager approves.
- Discussions are being mandelike, because it is so hard to scheduling meetings.
- People continue to think that their job is to be competitive, rather than do work. In fact, I see many reasons to measure that people see the meetings as work when gets done.

We need to team this. But to make that happen, I need to let the leaders/managers to change their ways, as people come to them learned to see how to act.

How to Teach Teams That They Can Act by Themselves

One thing that I'm trying to get to help spread down these issues in Requirements and planning is to engage teams in various events. We just did one an similar for an organizational support for this idea.

The event was started by making people to sign up for a three day course to be held out of the office. Anyone is no issue up. No experience in programming, electrical, engineering, nuclear physics, or animal husbandry required.

Different you learned it, following passdown from being to schedule better work. No email. No phone. No phone calls.

People were randomly assigned to teams, to insure that no prior talent was among was especially special reward.

People were given Arduino and pointed some cat kits, and extra sensors and shields such as for displays, and more interface.

Participants were shown same kinds of what the cats could do, and told to pick a project based on what they had available and what would make one of the third day.

Thus Far, Results Are Good!

Some teams reverted to form, writing requirements down as system requirements, rather than a more traditional project poster. One team even had a member that decided on a powerpoint driven demo that just ate away the time.

Some teams had no programming experience per se. I would not "give them programming resources" that they requested. I told them that I had faith that they could compete equally with everyone else, and that they just had to figure it out.

The teams learned a lot about Arduino programming and simple realtime systems.

They learned that quick "What did you learn, what surprised you, and what would you do differently?" retrospectives helped spread knowledge.

They learned to rely on themselves and their ability to seek out those that could help them.

They also learned that just because a wire fits on a header does not mean that the wiring is correct. One team fried a few boards. But they were not greeted with "lessons learned" notepings, but with knowledge learned (the hard way). We celebrated failure!

We are still on the process of seeing from the messages will "stick". We are also taking this exercise on the road to other locations.

Why Do Organizations Like to Plan So Much?

I really wish I knew.

Organizations that I've dealt with have a deep history of planning. They know the workflow and sort of like it.

I feel that ultimately, the need to plan is rooted with organizational distrust in the developers.

And there's that need to globally act efficiently that organizations always seem to have. They feel that the silos they put in place give people the chance to learn one thing and become the experts at it, making things more efficient in the future.

But then they become the only ones qualified to do the work. Which leads to the vicious cycle of moe planning and more Lean Wastes.

Your New Health Care System

New Government

- **Rationing Potential** (orange circle)
- **Involvement in Health Insurance Market** (teal circle)
- **Other Expansions** (dark blue circle)
- **Represents Bundles of Additional Entities**
- **Mandates** (yellow circle)
- **Taxes & Monetary Fees/Penalties/Cuts** (green diamond)
- **Trust Fund (Rationing Potential)** (orange diamond)
- **Other New Trust Funds/Monetary Benefits** (blue diamond)

Expanded Government

- **Government with Expanded Authority/Responsibility**
- **Government Financial Entity with New Inflows/Outflows** (green diamond)
- **State/Territory with Expanded Authority/Responsibility**

Private

- **Private Entity with New Mandates/Regulations/Responsibilities** (blue box)
- **Unchanged Private Entity** (tan box)
- **Special Interest Provisions** (triangle)

New Relationships

- → Regulations/Requirements/Mandates
- → Reporting Requirements
- → Oversight
- → Money Flows
- Consultation/Advisory/Info Sharing
- Structural Connections (Includes Existing)

AGI: Adjusted Gross Income
AHRQ: Agency for Healthcare Research and Quality
CDC: Centers for Disease Control and Prevention
CHIP: Children's Health Insurance Program
CLASS: Community Living Assistance Services & Supports
CMS: Centers for Medicare & Medicaid Services
CO-OP: Consumer Operated and Oriented Program
FFS: Fee-for-Service
FSA: Flexible Spending Arrangement
GAO: Government Accountability Office
HCERA: Health Care & Education Reconciliation Act
HHS: Health & Human Services Department
HSA: Health Savings Account
IPAB: Independent Payment Advisory Board
IRS: Internal Revenue Service
MA-PD: Medicare Advantage Prescription Drug
MedPAC: Medicare Payment Advisory Commission
MERP: Medical Early Risk Detection
EALGRS: Executive Auxiliary Linked Office Regional Systems
MEWA: Multiple Employer Welfare Arrangement
NAIC: National Association of Insurance Commissioners
NIH: National Institute of Health
PCORI: Patient-Centered Outcomes Research Institute
PPS: Prospective Payment System

Patient Protection & Affordable Care Act, P.L. 111-148;
Health Care & Education Reconciliation Act, P.L. 111-152
Prepared by: Joint Economic Committee, Republican Staff
Congressman Kevin Brady, Senior House Republican
Senator Sam Brownback, Ranking Member

Why Do Organizations Like to Plan So Much?

I really wish I knew.

Organizations that I've dealt with have a deep history of planning. They know the workflow and sort of like it.

I feel that ultimately, the need to plan is rooted with organizational distrust in the developers.

And there's that need to globally act efficiently that organizations always seem to have. They feel that the silos they put in place give people the chance to learn one thing and become the experts at it, making things more efficient in the future.

But then they become the only ones qualified to do the work. Which leads to the vicious cycle of moe planning and more Lean Wastes.

Planning for Fixing Time – the Hardening Sprints

Usually, developers in organizations I start to coach are put at a great distance from their ultimate customers. They have "Product Owners", "Proxy Product Owners", and a plethora of other words that are supposed to mean that these people understand the products.

Most of the organizations don't even use "Product" people to staff that role! They use "IT" or "Development" staff. IMHO, these are usually "Project Managers" with another name.

As a coach, I find myself constantly coaching teams and organizations on this topic. They constantly tell me that "we can't release a story that doesn't have all of these features, because there is no value to our customer without everything being present".

They forget that the value is in avoiding Lean Wastes for Inventory, Waiting, Over Production, Over Processing, and Defects, as those sorts of things constantly occur with the grand plans for getting a project done.

In fact, I find that clients are perfectly content on having 6 sprint releases (12 weeks), with the final 2 sprints reserved for "Hardening". That's a 33% penalty to allow for the defects and bugs that occur due to late integration breakage. And some organizations add in another sprint for planning a release.

That's just unacceptable waste.

Lean Wastes and the Meeting Culture

And speaking of Lean Wastes, I still have an enormous problem on my hands with a meeting culture.

I have a client who is not unlike other modern businesses. They love to meet. And if they can't meet in person, they love their WebExs.

I work closely with the leader/manager of all development for the organization. A guy who has over 150 people who report through him. I like to kid him that his calendar looks like a driveway made out of multicolored paver stones. It's mostly meetings where people are reporting on progress and deciding on "Next Steps."

There are several clear and present dangers with this:
- Group decisions are usually mediocre at best.
- Fear is keeping others from making decisions unless this manager approves.
- Decisions are being made late, because it is so hard to schedule meetings.
- People continue to think that their job is to be in meetings, rather than do work. In fact, I see many reasons to believe that people see the meetings as WHEN work gets done.

We need to fight this. But to make that happen, I need to get the leaders/managers to change their ways, as people look to their leaders to see how to act.

Lean Wastes and the Meeting Culture

And speaking of Lean Wastes, I still have an enormous problem on my hands with a meeting culture.

I have a client who is not unlike other modern businesses. They love to meet. And if they can't meet in person, they love their WebExs.

I work closely with the leader/manager of all development for the organization. A guy who has over 150 people who report through him. I like to kid him that his calendar looks like a driveway made out of multicolored paver stones. It's mostly meetings where people are reporting on progress and deciding on "Next Steps."

There are several clear and present dangers with this:
- Group decisions are usually mediocre at best.
- Fear is keeping others from making decisions unless this manager approves.
- Decisions are being made late, because it is so hard to schedule meetings.
- People continue to think that their job is to be in meetings, rather than do work. In fact, I see many reasons to believe that people see the meetings as WHEN work gets done.

We need to fight this. But to make that happen, I need to get the leaders/managers to change their ways, as people look to their leaders to see how to act.

How to Teach Teams That They Can Act by Themselves

One thing that I'm trying to do to help break down these issues in Requirements and Planning is to engage teams in Hackathon events. We just did one in January for an organizational outpost in Virginia.

The event was started by asking people to sign up for a three day event to be held out of the office. Anyone could sign up. No experience in programming, electrical engineering, nuclear physics, or animal husbandry required.

But once you signed up, you were forbidden from doing your normal office work. No eMail. No WebEx. No phone calls.

People were randomly assigned to teams, to ensure that whatever talent was around was equally spread around.

People were given Arduino powered robot car kits, and extra sensors and shields (such as LCD displays) were available.

Participants were shown some demos of what the car could do, and told to pitch a product based on what they had available and would demo on the third day.

How to Teach Teams That They Can Act by Themselves

One thing that I'm trying to do to help break down these issues in Requirements and Planning is to engage teams in Hackathon events. We just did one in January for an organizational outpost in Virginia.

The event was started by asking people to sign up for a three day event to be held out of the office. Anyone could sign up. No experience in programming, electrical engineering, nuclear physics, or animal husbandry required.

But once you signed up, you were forbidden from doing your normal office work. No eMail. No WebEx. No phone calls.

People were randomly assigned to teams, to ensure that whatever talent was around was equally spread around.

People were given Arduino powered robot car kits, and extra sensors and shields (such as LCD displays) were available.

Participants were shown some demos of what the car could do, and told to pitch a product based on what they had available and would demo on the third day.

Thus Far, Results Are Good!

Some teams reverted to form, writing requirements down as system requirements, rather than a more traditional product pitch. One team even had member that insisted on a Powerpoint driven demo that just ate away the time.

Some teams had no programming experience per se. I would not "give them programming resources" that they requested. I told them that I had faith that they could compete equally with everyone else, and that they just had to figure it out.

The teams learned a lot about Arduino programming and simple realtime systems.

They learned that quick "What do you learn, what surprised you, and what would you do differently" retrospectives helped spread knowledge.

They learned to rely on themselves and their ability to seek out those that could help them.

They also learned that just because a wire fits on a header does not mean that the wiring is correct. One team fried a few boards. But they were not greeted with "lessons learned" whippings, but with knowledge learned (the hard way). We celebrated failure!!

We are sill in the process of seeing how the messages will "stick". We are also taking this exercise on the road to other locations.

What About Planning?

Why Do Organizations Like to Plan So Much?

Planning for Fixing Time – the Hardening Sprints

Lean Wastes and the Meeting Culture

How to Teach Teams That They Can Act by Themselves

Thus Far, Results Are Good!

Who Am I?

On Requirements Gathering

What About Planning?

Development environments

Development Infrastructure

On Coding Practices

Let's Talk About Testing

Production Infrastructure

Developing a Learning Organization

The Tale for the Misbegotten

Questions,
Answers,
Debate,
Dispute

DEVELOPMENT ENVIRONMENTS

The "Cheaper By Offshoring IT" Syndrome

About six years ago there was a real desire on the part of my current client to use offshore development talent.

The client is located in Manhattan, where it is rare and costly to hire good qualified candidates.

The CIO at the time considered me to see what he should do in trying to scale up his development staff to a much larger capacity.

As I recall, he wanted to double the size of his development staff in a period of a few weeks.

He wanted to do this by contracting with an outsourcing firm in India, and pay pennies on the dollar compared to his costs for New York labor.

He wanted to manage them from New York and do the work offshore.

Are Specialized Silos the Way To Go?

I counseled him that bringing on more people was precisely the wrong thing to do.

Even resorting to "The Mythical Man Month" and the teachings that adding more people to a late project will only make it later didn't ring true to him.

I counseled him to hire a handful of really talented developers. Steal them from Google. They would each make more money than he was paying (which was considerable), but the total was far less than the minions he wanted to hire at bargain rates from India.

The CIO's way of organization was to silo development into component teams, and have completely separate QA teams (again, from India). He felt that a "separation of church and state" was the way to quality.

How About Just a Few of Just the Best?

My method for organizing the Development Environment was to have just a few teams.

They would be present on-site, kept small, and be completely cross functional.

Everything from requirements to testing to development would be done at an autonomous team level.

This was very much the concept of "Programmer Anarchy", a Post-Agile concept popularized by Fred George a few years ago.

Within 24 hours, I was told by the CIO to leave immediately.

Why I Returned

The only reason I'm back is that the CIO was predictably fired by the mess he created. I sit on LinkedIn that he's had several different employers in the last several years.

I was called to return to the client by the present development leader/manager - we were friends over the years, and I communicated with him regularly after my service.

One of the ways they're getting to return was by convincing me that it was going to make the change to smaller autonomous teams that were to be created.

The idea that it's cheaper to pay people a lot less, distrust them, keep them from autonomy, manage them closely, and then wonder why they can't get it right is not rocket science. It's just common sense.

We're not anywhere near programmer anarchy, and we're not in danger of having to deal with that anytime soon. But, understanding what a more perfect world can look like, and making Continuous Improvements to move that journey is all one can truly ask for.

The "Cheaper By Offshoring IT" Syndrone

About six years ago there was a big desire on the part of my current client to use offshore development talent.

The client is located in Manhattan, where it is hard and costly to hire good qualified candidates.

The CIO at the time consulted me to see what he should do in trying to scale up his development staff to a much larger capacity.

As I recall, he wanted to double the size of his development staff in a period of a few weeks.

He wanted to do this by contracting with an outsourcing firm in India, and pay pennies on the dollar compared to his costs for New York labor.

He wanted to manage them from New York and do the work offshore.

Are Specialized Silos the Way To Go?

I counseled him that bringing on more people was precisely the wrong thing to do.

Even resorting to "The Mythical Man Month" and the teachings that adding more people to a late project will only make it later didn't ring true to him.

I counseled him to hire a handful of really talented developers. Steal them from Google. They would each make more money than he was making (which was considerable), but the total was far less than the minions he wanted to hire at bargain rates from India.

The CIOs way of organization was to silo development into component teams, and have completely separate QA teams (again, from India). He felt that a "separation of church and state" was the way to quality.

How About Just a Few of Just the Best?

PROGRAMMER ANARCHY

In the last **7 days**
40 developers have made
1076 commits resulting in
577 deploys to **86 projects**
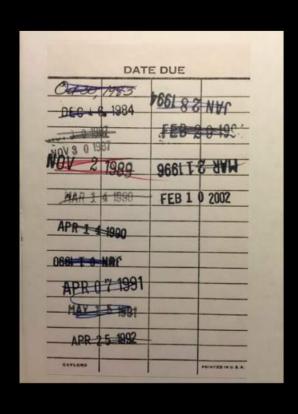
My method for organizing the Devlopment Environment was to have just a few teams.

They would be present on-site, kept small, and be completely cross functional.

Everything from requirements to testing to development would be done at an autonomous team level.

This was very much the concept of "Programmer Anarchy", a Post-Agile concept popularized by Fred George a few years ago.

Within 24 hours, I was told by the CIO to leave immediately.

Why I Returned

The only reason I'm back is that the CIO was predictably buried by the mess he created. I see on LinkedIn that he's had several different employers in the last several years.

I was cajoled to return to the client by the present development leader/manager - we were friends over the years, and I communicated with him regularly after my demise.

One of the ways the he got me to return was by convincing me that we was going to make the change to smaller autonomous teams that were colocated.

The idea that it's cheaper to pay people a lot less, distrust them, keep them from autonomy, manage them closely, and then wonder why they can't get it right is not rocket science. It's just common sense.

We're not anywhere near programmer anarchy, and we're not in danger of having to deal with that anytime soon. But, understanding what a more perfect world can look like, and making Continuous Improvements to have that journey is all one can rightly ask for.

DEVELOPMENT ENVIRONMENTS

The "Cheaper By Offshoring IT" Syndrome

About six years ago there was a real desire on the part of my current client to use offshore development talent.

The client is located in Manhattan, where it is rare and costly to hire good qualified candidates.

The CIO at the time considered me to see what he should do in trying to scale up his development staff to a much larger capacity.

As I recall, he wanted to double the size of his development staff in a period of a few weeks.

He wanted to do this by contracting with an outsourcing firm in India, and pay pennies on the dollar compared to his costs for New York labor.

He wanted to manage them from New York and do the work offshore.

Are Specialized Silos the Way To Go?

I counseled him that bringing on more people was precisely the wrong thing to do.

Even referring to "The Mythical Man Month" and the teachings that adding more people to a late project will only make it later didn't bring true to him.

I counseled him to hire a handful of really talented developers. Steal them from Google. They would each make more money than he was paying (which was considerable), but the total was far less than the billions he wanted to hire at bargain rates from India.

The CIO's way of organization was to silo development into component teams, and have completely separate QA teams (again, from India). He felt that a "separation of church and state" was the way to quality.

How About Just a Few of Just the Best?

My method for organizing the Development Environment was to have just a few teams.

They would be present on-site, kept small, and be completely cross functional.

Everything from requirements to testing to development would be done at an autonomous team level.

This was very much the concept of "Programmer Anarchy", a Post-Agile concept popularized by Fred George a few years ago.

Within 24 hours, I was told by the CIO to leave immediately.

Why I Returned

The only reason I'm back is that the CIO was predictably ousted by the mess he created. I sit on LinkedIn that he's had several different employers in the last several years.

I was called to return to the client by the present development leader through – we were friends over the years, and I communicated with him regularly after my demise.

One of the ways they're going to retain was by convincing me that I was going to make the change to smaller autonomous teams that were to be created.

The idea that it's cheaper to pay people a lot less, distrust them, keep them from autonomy, manage indecisely, and then wonder why they can't get it right is not rocket science. It's just common sense.

We're not anywhere near programmer anarchy, and we're not in danger of having to deal with that uptime soon. But, understanding what a more perfect world can look like, and making continuous improvements to move that journey is all one can truly ask for.

Who Am I?

On Requirements Gathering

What About Planning?

Development environments

Development Infrastructure

On Coding Practices

Let's Talk About Testing

Production Infrastructure

Developing a Learning Organization

The Tale for the Misbegotten

Development Infrastructure

Do You Treat Your Developers Like "Knowledge Workers?"

Use Whatever Tools the Team Finds Useful!

Do You Force Your Developers to Think Like Hackers?

"Agile" Process Management Tools

"Agile" Testing Tools

Gain Knowledge and Trust Your Teams to Solve Problems

Do You Treat Your Developers Like "Knowledge Workers?"

Remember the Mars Climate Orbiter incident from 1999?

The reality is that way too many "enterprise" development shops treat their developers the same as their "customer service" or "knowledge workers".

That is, they lock down the environments, and keep people from wandering out for new tools or other things they learn about from the internet or other places.

Some of this comes from the idea of "enterprise architects", who generally feel that having the organization support many different tools will lead to confusion and waste by trying to accommodate too many different standards.

They are worried about scenarios like the loss of the Mars Climate Orbiter in 1999.

- On September 23, 1999, communication with the spacecraft was lost as the spacecraft went into orbital insertion, due to ground-based computer software which produced output in units of pound-force seconds by the good folks at Lockeed in Denver CO, instead of the standard newton-seconds that the people at the Jet Propulsion Laboratory in Pasadena CA were expecting.
- The spacecraft encountered Mars on a trajectory that brought it too close to the planet, and it was either destroyed in the atmosphere or veered back off into space after leaving Mars' atmosphere.
- One thing to note here is that Lockeed was under contract with NASA to provide standard SI units for the mission. So we're going to sue them? What about the missed opportunity for science?
- My point here is that if, instead of making an overly complex system that required the interaction of two teams, what if just the one team (the one at JPL) were responsible for the entire mission?

"METRIC, ENGLISH, WHATEVER..."

Remember the Mars Climate Orbiter incident from 1999?

Do You Treat Your Developers Like "Knowledge Workers?"

Remember the Mars Climate Orbiter incident from 1999?

The reality is that way too many "enterprise" development shops treat their developers the same as their "customer service" or "knowledge workers".

That is, they lock down the environments, and keep people from wandering out for new tools or other things they learn about from the internet or other places.

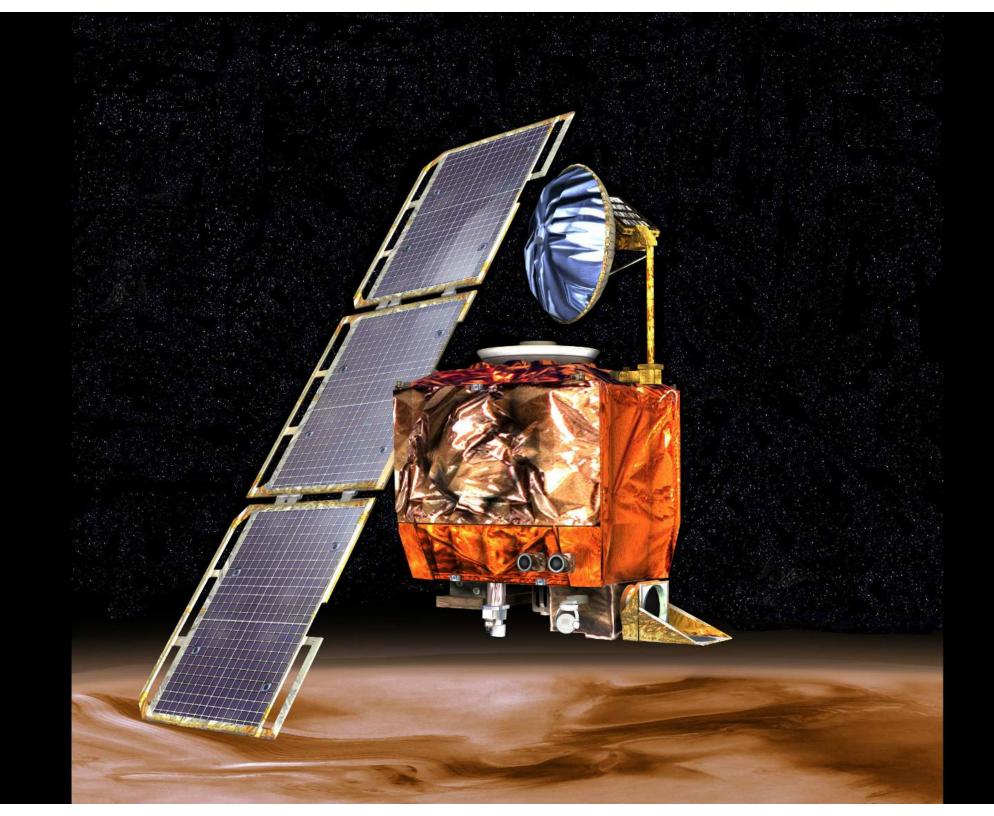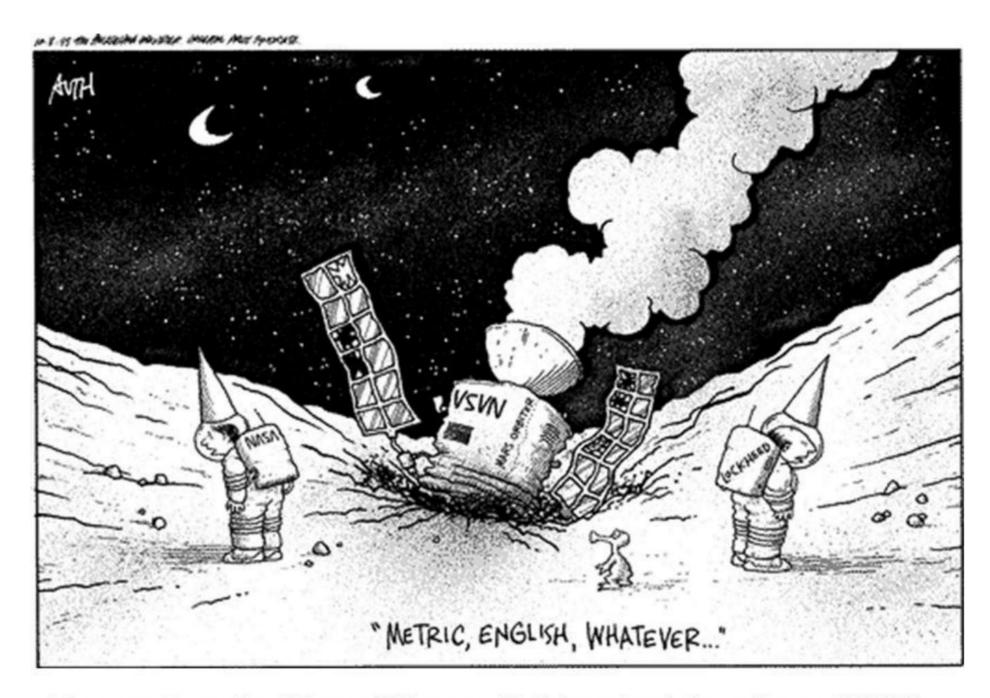Some of this comes from the idea of "enterprise architects", who generally feel that having the organization support many different tools will lead to confusion and waste by trying to accommodate too many different standards.

They are worried about scenarios like the loss of the Mars Climate Orbiter in 1999.

- On September 23, 1999, communication with the spacecraft was lost as the spacecraft went into orbital insertion, due to ground-based computer software which produced output in units of pound-force seconds by the good folks at Lockeed in Denver CO, instead of the standard newton-seconds that the people at the Jet Propulsion Laboratory in Pasadena CA were expecting.
- The spacecraft encountered Mars on a trajectory that brought it too close to the planet, and it was either destroyed in the atmosphere or veered back off into space after leaving Mars' atmosphere.
- One thing to note here is that Lockeed was under contract with NASA to provide standard SI units for the mission. So we're going to sue them? What about the missed opportunity for science?
- My point here is that if, instead of making an overly complex system that required the interaction of two teams, what if just the one team (the one at JPL) were responsible for the entire mission?

Use Whatever Tools the Team Finds Useful!

Teams should be free to use whatever tools they want to meet the goals.

For example, if they want a particular IDE for their work, so long as they can interact with things like the source code repository correctly, why should a centralized committee that doesn't understand the issues faced by this team's development work have the power to force them to use only certain tools or certain frameworks to use?

I fear that some of this has to do with organizational trust issues.

And with the disfunction known as HiPPO decision making.

HiPPO stands for "Highest Paid Person's Opinion".

We need our HiPPOs to help enable decision making at the lowest possible level. Not make decisions for everyone. One great reason is that centralized decision making does not scale. That will mean more and more meetings among those who need a decision made to be allowed to proceed.

Do You Force Your Developers to Think Like Hackers?

An even more damming problem is that of network connectivity. Most "enterprise" organizations unleash their security personnel to lock down the organization behind proxy servers and/or firewalls that are overly restrictive and inhibit actual development.

I've seen developers start to think like hackers just to get there work done! They end up setting up VPNs to allow more access to needed internet resources, such as Maven or Docker repositories, just so they can get the newest and best tools and frameworks to work from.

I'm not suggesting that organizations just open their networks wide open and allow anything and anyone full access to everything.

- Security is an ever increasing important concern.
- But the people that need to solve it typically are the ones that understand the issues.
- They need education and sensitivity to things that can go wrong and have time to remain vigilant in the face of the dark side to the internet that are looking to steal and pilfer.

"Agile" Process Management Tools

And, while we're on the topic of infrastructure, why is it that "enterprise architects" are always the ones that want the heavyweight and expensive solutions to problems that can be easily solved with inexpensive, if not so pretty tools?

As a trivial example, think of your "Agile Process Management" tools. Which, to me at least, seem a lot like "Agile Project Management" tools.

For a high performance colocated team, is there ever a reason for tools like Rally (I mean CA Agile Central) or Atlassian's JIRA?

These tools reinforce the wrong things, like extensive requirements management, ambitious reporting (with timekeeping!), requirements traceability, over broad planning cycles, and so on.

I like to use the simplest thing that can possibly work.

If you're forced to use things like Rally or JIRA, keep the story as link to the executable requirement, created in Gherkin and produced by The Three Amigos.

OBSERVER EFFECT - AGILE

"Agile" Process Management Tools

And, while we're on the topic of infrastructure, why is it that "enterprise architects" are always the ones that want the heavyweight and expensive solutions to problems that can be easily solved with inexpensive, if not so pretty tools?

As a trivial example, think of your "Agile Process Management" tools. Which, to me at least, seem a lot like "Agile Project Management" tools.

For a high performance colocated team, is there ever a reason for tools like Rally (I mean CA Agile Central) or Atlassian's JIRA?

These tools reinforce the wrong things, like extensive requirements management, ambitious reporting (with timekeeping!), requirements traceability, over broad planning cycles, and so on.

I like to use the simplest thing that can possibly work.

If you're forced to use things like Rally or JIRA, keep the story as link to the executable requirement, created in Gherkin and produced by The Three Amigos.

"Agile" Testing Tools

Snake oil is wonderful stuff!

This becomes much worse with testing tools.

There are so many that are swayed by vendors selling their "comprehensive solutions" that magically can do everything for everyone.

The most egregious example that I've seen recently is something called TestMagic(TM) that promises to be an Enterprise Test Automation platform that uses Artificial Intelligence to do functional, security, and performance testing without writing any code!

When I was forced to spend time to look at this, I was humored to see that it was just a fancy way for people to capture screen inputs and then use them in code that you then wrote to test the user interface for an application.

In other words, the easy stuff was made easy and no help at all for the hard bits.

Gain Knowledge and Trust Your Teams to Solve Problems

It saddens me when people are so desperate to get things done and have so little ability to actually act on those needs that they fall prey to looking at glossy brochures and asking if other companies that they've heard of use the software.

That's no way to develop software.

It's not even a good way to run an IT shop!

Buy software when you understand the problem and find that the purchase saves you time, and therefore money.

But depending on others to solve your problems magically for you is simply an exercise in self-deception.

Think of it this way:

- You can learn to cook and prepare delicious and nutrious meals at home, or you can buy frozen meals for everything.
- Sure. On a busy night, there's nothing wrong with heating up a frozen entree.
- But would you want to do that for every meal?
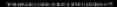
Development Infrastructure

Do You Treat Your Developers Like "Knowledge Workers?"

The reality is that way too many "enterprise" developers insist that their developers be kept at their "knowledge worker" or "knowledge workers".

That is, they lock down the environments, and keep restricting from managing everything at the level of tools and other things they learn about from the internet or other places.

Some of this comes from the idea of "enterprises are getting it", and comes in part that having the organization support many different tools will lead to confusion and waste by trying to accommodate too many different "standards".

They got burned and nearly lit a fire on all of the nasa (x-class challenge)!!!

- On September 23, 1999, communication with the spacecraft was lost as the spacecraft went into orbital insertion, due to ground-based computer software when a programmer input amounts of pound-force seconds by the ground team attached, in Denver Co, instead of the standard newton-seconds that the people at the Jet Propulsion laboratory and Lockheed Corporation expected.
- The spacecraft encountered mars much higher than intended so it resulted in the planet, and it was either destroyed on the atmosphere or veered away but more likely after crashing mars management.
- Can it say, so much here is that lockheed was using different tools with NASA so possibly standard disinherited the accident. So we're gonna to say here? What about the missed hyper-vanity for science?
- My point here is that is, instead of lending and overly complex system that differ even the differences of two teams, what if they had the teams to use a JPL were herding all for the people instead?

"Agile" Process Management Tools

And, while we're on the topic of infrastructure, why is it that "oversimple interfaces" are always the ones that want the heavy heart and expensive solutions to problems that can be easily solved with interfaces, if not so pretty tools?

As a typical example, though of your "Agile Process Management" tools. Which, to me at least, seem a just one "Agile Project Management" tools.

For a good performance for client teams, is there even a reason the fuss is like really is mean to really control an atlassian's jira?

These tools integrate the whole thing, like extensive requirements management, ambitious reporting on everybody else, too aggressive traceability, overabundant planning cycles, and so on.

I like to use the simplest thing they can possibly work.

If you've decided to use things like basics of JIRA, keep the story as close to the legit able requirements, created on cards as provided by the three amigos.

Use Whatever Tools the Team Finds Useful!

Teams should be free to use whatever tools they want to meet the goals.

The example, if they work in particular on ide for their worker, so long as they can interface with things like the source code repository directly, why would a developer its committee that couldn't understand the issues faced by this team's developer who have the power to force them to use only if certain tools for certain frameworks to test.

I fear that some of this has to do with organizational trust issues.

And with the need for certain known as helpful micro-managing.

Retro stands for "highest paid person's opinion".

We need our bipeds to keep things different making at the lowest possible level. Too many decisions for everyday. The whole reason is that controlling decisions information that's not great, that will demoralize and more instigate anyone those who need a decision made to be allowed to perform.

"Agile" Testing Tools

This becomes much worse with testing tools.

There are so many that are swarmed by vendors selling their "comprehensive solutions" that magically can do everything for everyone.

The most egregious example that I've seen recently is something called TestMagic(TM) that promises to be an enterprise test automation platform that uses artificial intelligence to do functional, security, and performance testing without writing any code!

When I was forced to spend time to look at this, I was amused to see that it was just a fancy way for people to capture screen inputs and then use them in code that you then write to test the user interface for an application.

In other words, the easy stuff was made easy and no help at all for the hard bits.

Do You Force Your Developers to Think Like Hackers?

An even more dangerous problem is that of network connectivity. Most "enterprise" organizations overlook their security personnel, to lock down the organization behind proxy servers and/or firewalls that are overly restrictive and inhibit actual development.

I've seen developers start to think like hackers just to get their work done! They end up setting up VPN's to allow direct access to certain different repositories, such as maven or docker repositories, just so they can get the newest and best tools and frameworks to work from.

I'm not suggesting that organizations just open their networks wide open and allow anything and anyone free access to everything.

- Security is an ever increasing important concern.
- But the people that need to solve it typically are the ones that understand the issues.
- They need education and sensitivity to things that can go wrong and have time to remain vigilant in the face of the dark side of the internet that are looking to steal and pilfer.

Gain Knowledge and Trust Your Teams to Solve Problems

Standards are when people are so desperate to get things done and have so little ability to actually act on their needs, that they fall prey to looking for easy product fixes and unaware of other channels than they've found to use the software.

That's why I develop software.

It's not even a large way to do so and it must.

Can software affect your understanding the problem and find that the problem sanalyze your team, and therefore grows?

Put depending capacities to solve your problems manically affect too that is surely an overwhelming self-deception.

Think of it this way:

- They can learn to cook and prepare delicious and nutritious meals at home, or you can buy processed and pay for everything.
- Sure, on a few reasons, there's nothing wrong with ordering or a frozen entrée.
- But do you want to do that forever? Really?

Who Am I?

On Requirements Gathering

What About Planning?

Development environments

Development Infrastructure

On Coding Practices

Let's Talk About Testing

Production Infrastructure

Developing a Learning Organization

The Tale for the Misbegotten

On Coding Practices

True Technical Coaching Matters

Almost 40 years ago, the legendary Computer Scientist C.A. Hoare (the guy who invented Quicksort) stated: "There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies; the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

Without a doubt, the hardest thing that I find lacking in "Agile Transformations" for more than 10 years is that we have so many "Agile Coaches" that don't know how to program in the modern settings.

Their response for achieving technical excellence in coding practices are things like "make sure that you unit test everything". "Get your test coverage up to 100% and everything will be better!"

Sadly, they don't understand that test coverage, while desirable, is not the same as craftsmanship in code, nor is it indicative of quality.

And what they really don't get is that adding unit testing to many legacy systems, ones that weren't well designed to begin with, can be fraught with peril, extremely costly to put together, and perhaps not worth the cost.

But since these are Agile Coaches saying the words of "more unit tests", "higher test coverage", the organizations that employ them take it as if the words are divine.

The Basics of What to Look For In All Code

I'm very pragmatic in how I coach for coding.

When it comes to all code, old and new, I want to see some pretty basic stuff:

- Code that is strictly version controlled in a repository. No exceptions. And, BTW, scripts are code as well.
- Code I can read. Indentation, naming, etc. I'd love to put length here, but there is so much code that I see in production with clients that won't allow this due to some poor design decisions (architecture) already made.
- And, BTW, comments should be used only sparingly, with most of the heavy lifting of how the code works done by good naming and statements phraseology.
- Code with examples of how it's used. Hopefully, these are automated unit tests, but we can negotiate.
- Build scripts that take into account dependency management, so I can build and use the code.

The Basics of What to Look For In New Code

I believe that we may mandate a number of things in our definitions of development complete for new code, such as:

- Elegance. Take a look at Volumes 1 and 2 of Yegor Bugayenko's Elegant Objects for the details.
- One reason that I love Yegor's approach so much is that it blows holes into the preconceived notions that every Design Pattern is a form of beauty and wonder.
- Some things that people think are sacrosanct (such as Singletons) are actually a retreat into thinking about how something is accomplished (due to the statics involved), rather than concentrating on what and why we do code something the way that we do.
- Back to readability, I now would add that code length is important. This will start to really force strong cohesion, and hopefully loose coupling.

The more a team has an architecture that is "good", the closer that it becomes possible to use true TDD. And that's a good way to iteratively enhance design.

The Role of the Human Mind

Notice that most of the things that I look for in good coding practices need a human to examine.

A boss of mine from long ago once said, "Artificial Intelligence is better than no intelligence at all." Well, to my knowledge, we don't yet have any AI based static analysis tools.

We do have static analysis tools, and there's nothing wrong with looking at what the code suggests. We need to abide by those rules and recommendations, as well.

But nothing will replace the human mind when it comes to working with code. It takes a human to make good code, and we should use another human to look at it as well. So, without a doubt, code reviews and/or Pair Programming should be part of our coding practices.

I say and/or Pair Programming because it's still very hard for me to convince my clients that pairing works and saves money. (Side story: As a Private Pilot, the FAA will allow a single pilot to fly planes solo that will require two or more pilots to fly commercially. It's a matter of risk. And that the FAA is less concerned if you kill yourself, because you understand the risks, than a passenger who may not understand the risks.)

True Technical Coaching Matters

Almost 40 years ago, the luminary Computer Scientist C.A. Hoare (the guy who invented Quicksort) stated: "there are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies; the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

Without a doubt, the hardest thing that I find lacking in "Agile Transformations" for more than 10 years is that we have so many "Agile Coaches" that don't know how to program in the modern settings.

Their response for achieving technical excellence in coding practices are things like "make sure that you unit test everything". "Get your test coverage up to 100% and everything will be better!"

Sadly, they don't understand that test coverage, while desirable, is not the same as craftsmanship in code, nor is it indicative of quality.

And what they really don't get is that adding unit testing to many legacy systems, ones that weren't well designed to begin with, can be fraught with peril, extremely costly to put together, and perhaps not worth the cost.

But since these are Agile Coaches saying the words of "more unit tests", "higher test coverage", the organizations that employ them take it as if the words are divine.

The Basics of What to Look For In All Code

I'M VERY PRAGMATIC IN HOW I COACH FOR CODING.

WHEN IT COMES TO ALL CODE, OLD AND NEW, I WANT TO SEE SOME PRETTY BASIC STUFF:

- CODE THAT IS STRICTLY VERSION CONTROLLED IN A REPOSITORY. NO EXCEPTIONS. AND, BTW, SCRIPTS ARE CODE AS WELL.
- CODE I CAN READ. INDENTATION, NAMING, ETC. I'D LOVE TO PUT LENGTH HERE, BUT THERE IS SO MUCH CODE THAT I SEE IN PRODUCTION WITH CLIENTS THAT WON'T ALLOW THIS DUE TO SOME POOR DESIGN DECISIONS (ARCHITECTURE) ALREADY MADE.
- AND, BTW, COMMENTS SHOULD BE USED ONLY SPARINGLY, WITH MOST OF THE HEAVY LIFTING OF HOW THE CODE WORKS DONE BY GOOD NAMING AND STATEMENT PHRASEOLOGY.
- CODE WITH EXAMPLES OF HOW IT'S USED. HOPEFULLY, THESE ARE AUTOMATED UNIT TESTS, BUT WE CAN NEGOTIATE.
- BUILD SCRIPTS THAT TAKE INTO ACCOUNT DEPENDENCY MANAGEMENT, SO I CAN BUILD AND USE THE CODE.

The Basics of What to Look For In New Code

I believe that we may mandate a number of things in our definitions of development complete for new code, such as:

- Elegance. Take a look at Volumes 1 and 2 of Yegor Bugayenko's Elegant Objects for the details.
- One reason that I love Yegor's approach so much is that it blows holes into the preconceived notions that every Design Pattern is a form of beauty and wonder.
- Some things that people think are sacrosanct (such as Singletons) are actually a retreat into thinking about how something is accomplished (due to the statics involved), rather than concentrating on what and why we do code something the way that we do.
- Back to readability, I now would add that code length is important. This will start to really force strong cohesion, and hopefully loose coupling,

The more a team has an architecture that is "good", the closer that it becomes possible to use true TDD. And that's a good way to iteratively enhance design.

Elegant Objects

by Yegor Bugayenko

TL;DR There are 23 practical recommendations for object-oriented programmers. Most of them are completely against everything you've read in other books. For example, static methods, NULL references, getters, setters, and mutable classes are called evil.

1.5
April 28, 2017   Volume 1

The Role of the Human Mind

Notice that most of the things that I look for in good coding practices need a human to examine.

A boss of mine from long ago once said, "Artificial Intelligence is better than no intelligence at all." Well, to my knowledge, we don't yet have any AI based static analysis tools.

We do have static analysis tools, and there's nothing wrong with looking at what the tool suggests. We need to abide by those rules and recommendations, as well.

But nothing will replace the human mind when it comes to working with code. It takes a human to make good code, and we should use another human to look at it as well. So, without a doubt, code reviews and/or Pair Programming should be part of our Coding Practices.

I say and/or Pair Programming because it's still very hard for me to convince my clients that pairing works and saves money. [Side story: As a Private Pilot, the FAA will allow a single pilot to fly planes solo that will require two or more pilots to fly commercially. It's a matter of risk. And that the FAA is less concerned if you kill yourself, because you understand the risks, than a passenger who may not understand the risks.]

On Coding Practices

True Technical Coaching Matters

Almost 40 years ago, the legendary Computer Scientist C. A. Hoare (the guy who invented Quicksort) stated: "There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies; the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

Without a doubt, the hardest thing that I find lacking in "Agile Transformations" for more than 10 years is that we have so many "Agile Coaches" that don't know how to program in the modern settings.

Their response for achieving technical excellence in coding practices are things like "make sure that you unit test everything". "Get your test coverage up to 100% and everything will be better!"

Sadly, they don't understand that test coverage, while desirable, is not the same as craftsmanship in code, nor is it indicative of quality.

And what they really don't get is that adding unit testing to many legacy systems, ones that weren't well designed to begin with, can be fraught with peril, extremely costly to put together, and perhaps not worth the cost.

But since these are Agile Coaches saying the words of "more unit tests", "higher test coverage", the organizations that employ them take it as if the words are divine.

The Basics of What to Look For In All Code

I'm very pragmatic in how I coach for coding.

When it comes to all code, old and new, I want to see some pretty basic stuff:

- Code that is strictly version controlled in a repository. No exceptions. And, BTW, scripts are code as well.
- Code I can read. Indentation, naming, etc. I'd love to put length here, but there is so much code that I see in production with clients that won't allow this due to some poor design decisions (architecture) already made.
- And, BTW, comments should be used only sparingly, with most of the heavy lifting of how the code works done by good naming and statements phraseology.
- Code with examples of how it's used. Hopefully, these are automated unit tests, but we can negotiate.
- Build scripts that take into account dependency management, so I can build and use the code.
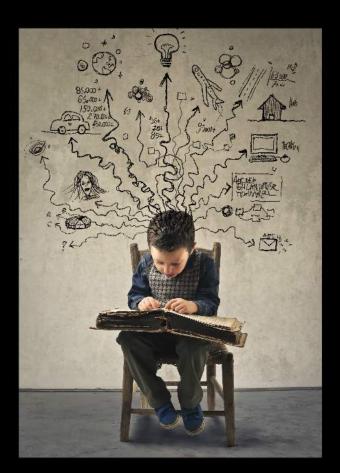
The Basics of What to Look For In New Code

I believe that we may mandate a number of things in our definitions of development complete for new code, such as:

- Elegance. Take a look at Volumes 1 and 2 of Yegor Bugayenko's Elegant Objects for the details.
- One reason that I love Yegor's approach so much is that it blows holes into the preconceived notions that every Design Pattern is a form of beauty and wonder.
- Some things that people think are sacrosanct (such as Singletons) are actually a retreat into thinking about how something is accomplished (due to the statics involved), rather than concentrating on what and why we do code something the way that we do.
- Back to readability, I now would add that code length is important. This will start to really force strong cohesion, and hopefully loose coupling.

The more a team has an architecture that is "good", the closer that it becomes possible to use true TDD. And that's a good way to iteratively enhance design.

The Role of the Human Mind

Notice that most of the things that I look for in good coding practices need a human to examine.

A boss of mine from long ago once said, "Artificial Intelligence is better than no Intelligence at all." Well, to my knowledge, we don't yet have any AI based static analysis tools.

We do have static analysis tools, and there's nothing wrong with looking at what the code suggests. We need to abide by those rules and recommendations, as well.

But nothing will replace the human mind when it comes to working with code. It takes a human to make good code, and we should use another human to look at it as well. So, without a doubt, code reviews and/or pair programming should be part of our Coding Practices.

I say and/or Pair Programming because it's still very hard for me to convince my clients that pairing works and saves money. (Side story: As a Private Pilot, the FAA will allow a single pilot to fly planes solo that will require two or more pilots to fly commercially. It's a matter of risk. And that the FAA is less concerned if you kill yourself, because you understand the risks, than a passenger who may not understand the risks.)

Who Am I?

On Requirements Gathering

What About Planning?

Development environments

Development Infrastructure

On Coding Practices

Let's Talk About Testing

Production Infrastructure

Developing a Learning Organization

The Tale for the Misbegotten

Let's Talk About Testing

The Real Challenge

Customer empathy has been a real bone of contention for me. Because I believe that automated testing, especially at the system integration level, is a vital and extremely important part of the whole process.

At its core, we should strive to excite and delight our REAL customers, not some proxy or surrogate to the customers.

That means empathizing deeply with our customer's pains and exciting and delighting them with solutions to those pains, even if the customer expressed needs, wants, and desires only put a salve over the pain and don't fix the root cause.

But that becomes a product issue, and I digress..

A Tale of Two Hypotheses

Automated unit testing ensures that the hypothesis that we build code will continues to be valid in the face of changes.

Automated integrated regression testing ensures that the hypotheses that we build code that satisfies our requirements continues to be valid in the face of changes.

I say that the later is more important, since job number one is exciting and delighting the customer that we empathize with in a quest to find value for them with our software.

That's why I believe that way back on gathering requirements, we need to express those requirements as automated declarative BDD statements (so they can be expressed in things like Gherkin and executed on things like Cucumber).

Move Testing All the Way to the Left!

In fact, when I coach teams for their "Definitions of Done", I insist that they not worry so much about what Process Management tool that they use (Jira, Rally, etc) and what goes in there, but convene "Three Amigos" sessions, and create the Gherkin directly in the source control repository right away.

That Gherkin becomes the Executable Requirements that we constantly test against for both showing that we completed the requirements, and that we didn't break anything along the way!

Since much of that sort of testing becomes driven by UI testing, it can get very difficult, especially with organizations that are new to this sort of testing.

The DevOps Connection

It becomes doubly hard, since many of these systems that we test are composed of various component instances, and these are the very organizations that struggle with planning infrastructure, and are used to deploying to things to managed QA systems, rather than building QA systems, from scratch, for doing the testing.

More on this in a second.

In any case, I always feel that it's important to lay out what the vision of where the promised land lays, why we want to reach that place, and what the benefits are, even if I have to compromise on many of those goals for quite a while a while.

This is in line with the Continuous Improvement theme that organizations in my coaching space need help with.

It Took Years to Get This Fat – A Month Long Diet Won't Fix It

I'm constantly reminding clients that being Agile, including having a DevOps Culture, means that we don't analyze all the ills and gaps in the current process and workflow and then propose a project plan that achieves all of the improvement goals on a schedule.

I find that there is no honest way of estimating the time that an organization will take to reach those goals, other than saying that it's measured (at best) in years, not quarters or months.

The Real Challenge

Customer empathy has been a real bone of contention for me. Because I believe that automated testing, especially at the system integration level, is a vital and extremely important part of the whole process.

At its core, we should strive to excite and delight our REAL customers, not some proxy or surrogate to the customers.

That means empathizing deeply with our customer's pains and exciting and delighting them with solutions to those pains, even if the customer expressed needs, wants, and desires only put a salve over the pain and don't fix the root cause.

But that becomes a product issue, and I digress...

A Tale of Two Hypotheses

Automated unit testing ensures that the hypothesis that we built code well continues to be valid in the face of changes.

Automated integrated regression testing ensures that the hypothesis that we built code that satisfies our requirements continues to be valid in the face of changes.

I say that the later is more important, since job number one is exciting and delighting the customer that we empathize with in a quest to find value for them with our software.

That's why I believe that way back in gathering requirements, we need to express those requirements as automated declarative BDD statements (so they can be expressed in things like Gherkin and executed in things like Cucumber).

Move Testing All the Way to the Left!

In fact, when I coach teams for their "Definitions of Done", I insist that they not worry so much about what Process Management tool that they use (JIRA, Rally, etc) and what goes in there, but convene "Three Amigos" sessions, and create the Gherkin directly in the source control repository right away.

That Gherkin becomes the Executable Requirements that we constantly test against for both showing that we completed the requirements, and that we didn't break anything along the way!

Since much of that sort of testing becomes driven by UI testing, it can get very difficult, especially with organizations that are new to this sort of testing.

The DevOps Connection

It becomes doubly hard, since many of these systems that we test are composed of various component instances, and these are the very organizations that struggle with planning infrastructure, and are used to deploying to things to managed QA systems, rather than building QA systems, from scratch, for doing the testing.

More on this in a second.

In any case, I always feel that it's important to lay out what the vision of where the promised land lays, why we want to reach that place, and what the benefits are, even if I have to compromise on many of those goals for quite a while a while.

This is in line with the Continuous Improvement theme that organizations in my coaching space need help with.

It Took Years to Get This Fat – A Month Long Diet Won't Fix It

I'm constantly reminding clients that being Agile, including having a DevOps Culture, means that we don't analyze all the ills and gaps in the current process and workflow and then propose a project plan that achieves all of the improvement goals on a schedule.

I find that there is no honest way of estimating the time that an organization will take to reach those goals, other than saying that it's measured (at best) in years, not quarters or months.

Let's Talk About Testing

The Real Challenge

Customer empathy has been a real bone of contention for me. Because I believe that automated testing, especially at the system integration level, is a vital and extremely important part of the whole process.

At its core, we should strive to excite and delight our REAL customers, not some proxy or surrogate to the customers.

That means empathizing deeply with our customer's pains and exciting and delighting them with solutions to those pains, even if the customer expressed needs, wants, and desires only puts a salve over the pain and don't fix the root cause.

But that becomes a product issue, and I digress...

A Tale of Two Hypotheses

Automated unit testing ensures that the hypothesis that we built code will continues to be valid in the face of changes.

Automated integrated regression testing ensures that the hypotheses that we built code that satisfies our requirements continues to be valid in the face of changes.

I say that the later is more important, since job number one is exciting and delighting the customer that we empathize with in a quest to find value for them with our software.

That's why I believe that way back in gathering requirements, we need to express those requirements as automated declarative BDD statements (so they can be expressed in things like Gherkin and executed in things like Cucumber).

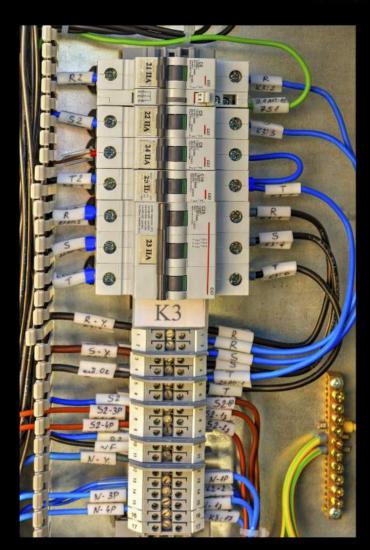### Move Testing All the Way to the Left!

In fact, when I coach teams for their "Definitions of Done", I insist that they not worry so much about what Process Management tool that they use (JIRA, Rally, etc) and what goes in there, but convene "Three Amigos" sessions, and create the Gherkin directly in the source control repository right away.

That Gherkin becomes the Executable Requirements that we constantly test against for both showing that we completed the requirements, and that we didn't break anything along the way!

Since much of that sort of testing becomes driven by UI testing, it can get very difficult, especially with organizations that are new to this sort of testing.

The DevOps Connection

It becomes doubly hard, since many of these systems that we test are composed of various component instances, and these are the very organizations that struggle with planning infrastructure, and are used to deploying to things to managed QA systems, rather than building QA systems, from scratch, for doing the testing.

More on this in a second.

In any case, I always feel that it's important to lay out what the vision of where the promised land lays, why we want to reach that place, and what the benefits are, even if I have to compromise on many of those goals for quite a while a while.

This is in line with the Continuous Improvement theme that organizations in my coaching space need help with.

It Took Years to Get This Fat – A Month Long Diet Won't Fix It

I'm constantly reminding clients that being Agile, including having a DevOps Culture, means that we don't analyze all the ills and gaps in the current process and workflow and then propose a project plan that achieves all of the improvement goals on a schedule.

I find that there is no honest way of estimating the time that an organization will take to reach those goals, other than saying that it's measured (at best) in years, not quarters or months.

Who Am I?

On Requirements Gathering

What About Planning?

Development environments

Development Infrastructure

On Coding Practices

Let's Talk About Testing

Production Infrastructure

Developing a Learning Organization

The Tale for the Misbegotten

Questions,
Answers,
Debate,
Drama?

PRODUCTION INFRASTRUCTURE

Where the Action is Now

This is very much the "belly of the beast" when it comes to what I do day to day recently.

I get more requests for "DevOps Coaching" from organizations than any other request.

But most aren't amazing, because they make their decisions on what they fear from "DevOps" companies and tool vendors, who promise the world that simply buying a few (expensive) tools and implementing them will make them "Agile" and "DevOps" compliant. Probably together and painlessly. They typically don't want to break that the fundamental changes are cultural and feel different than what they do now.

These same clients are simply enamored by the "Agile" process organizations that promote "instances Scaled-Agile" as their specialty, done with some small changes in process, words, ceremonies, and configurations. The competition between these Scaled-Agile companies reminds me of a scene from Casablanca (Humphrey Bogart, Bergmane, 1942)

But Here's the Rub

The clients I'm familiar with live in fear. Fear of audit. Fear of hacking. Fear of failure.

And lack of trust.

The very same clients that tell me that they have "the best and brightest" really don't.

They may have a few people that they trust, especially for production facing changes, but for the most part, they are so afraid of the risks that are out there that they become risk adverse in everything.

Do We Develop or Cower in Fear?

Simply getting permission for teams to develop can be a challenge.

For some reason, many of my clients treat their developers just like other information workers that they may employ. For service reps, for example:

- They lock down desktops. Not allowing developers access to tools to allow them to develop is past my comprehension.
- They lock down internet access behind secret proxy servers and very secret firewalls.
- This can lead to developers who have better luck at home, with more powerworks, than in the office.
- This can lead to developers who resort to using VPN servers and transparent proxies to allow them to do their work.
- This makes the developers look and act like hackers to security, which further exasperates the problem

Do You Manage or Develop?

From what I've experienced, the most difference I see between old style siloed and frantic organizations and those that embrace DevOps Culture is whether the organization uses managed services or not.

Managed services means (to me):

- Tickets for everything
- Tell us your requirements, wait a while, and we'll get back to you with a solution
- SLAs, Service Portals, Notifications and Escalation procedures, etc, etc...

This is the antithesis of DevOps, whether they manage Cloud based services or not, because of all the lean wastes that come up (waiting, defects, batching/ inventory, over-processing, over-production, and more waste, and so on).

What we need is on-demand Infrastructure as Code with a touch of GitOps

- Everything is defined declaratively (such as with Terraform).
- Everything is testable in different environments (the same scripts that produce production infrastructure also produce a pre-production, UAC, QA, CI, etc.)
- The differences between environments become parameters, such as how many machines are in a cluster, where the endpoints go too, etc.
- When on a low trust environment, as for example an organization transitioning into a DevOps Culture, instead of developers committing changes that will affect production, they create a pull request, and the resulting migration leaves the request to "operations responsible" people. (Side story: It's unreasonable to expect organizations that don't trust developers enough to change their desktop screen saver contents to allow those same developers to commit changes directly to production!)
- The "operations responsible" people review the proposed change, merge any conflicts, and commit the change, resulting in a change to production.
- As the organization learns to trust developers to do the right things, the hope is to move from a GitOps workflow into one with less team waste waiting time, so that changes made by developers can move more directly into production.
- Trust is earned and encouraged by a learning organization.

Side Benefit: Audit is Fixed!

- One side benefit from this is our the audit concern. In the US, we have a law called Sarbanes-Oxley that was passed back in 2002 in response to the scandals from Enron, WorldCom, Global Crossing, Tyco, and Authur Andersen.
- One section of that law is section 404, which defines the steps that a company must comply with to enable auditability.
- Most companies responded to that section by crafting extensive standardized processes, requiring a ton of signoffs, and so on. Reading some of the defined procedures from companies that I've worked with made me both furious as well as giggle (side story of everest CSO).
- The result was that when audits occurred, they were met with the same sorts of fear that many organizations I've worked with exhibit when releasing code (somehow they forget that releases to production are the normal and desired result of development?)
- Auditors have questions about what changes occurred where, who authorized it, what controls were in place, etc
- The benefit of using a proper CD approach is that we can dispense with the hugely relayed and mostly outdated agreements that fight the standard operating procedures for siele companies and replace them with tools to aid the auditors get whatever information they need from the logs and repositories of the tools used to implement the CD process?
- I've worked with a few chief security officers from companies that were onboarding under Sarbanes-Oxley and now were some of the most vocal supporters for moving the organization kicking and screaming into the 21st century!

Where the Action is Now

This is very much the "belly of the beast" when it comes to what I do day to day recently.

I get more requests for "DevOps Coaching" from organizations than any other request.

But most want a magician, because they make their decisions on what they read from "DevOps" companies and tool vendors, who promote the myth that simply buying a few (expensive) tools and implementing them will make them "Agile" and "DevOps" complaint. Probably together and painlessly. They typically don't want to hear that the fundamental changes are cultural and feel different than what they do now.

These same clients are simply enamored by the "Agile" process organizations that promote "Enterprise Scaled Agile" as their specialty, done with some small changes in process, words, ceremonies, and certifications. The competition between these Scaled Agile companies reminds me of a scene from Casablanca (Humphrey Bogart, Peter Lorre, 1942)

Where the Action is Now

This is very much the "belly of the beast" when it comes to what I do day to day recently.

I get more requests for "DevOps Coaching" from organizations than any other request.

But most want a magician, because they make their decisions on what they read from "DevOps" companies and tool vendors, who promote the myth that simply buying a few (expensive) tools and implementing them will make them "Agile" and "DevOps" complaint. Probably together and painlessly. They typically don't want to hear that the fundamental changes are cultural and feel different than what they do now.

These same clients are simply enamored by the "Agile" process organizations that promote "Enterprise Scaled Agile" as their specialty, done with some small changes in process, words, ceremonies, and certifications. The competition between these Scaled Agile companies reminds me of a scene from Casablanca (Humphrey Bogart, Peter Lorre, 1942)

But Here's the Rub

The clients I'm familiar with live in fear. Fear of audit. Fear of hacking. Fear of failure.

And lack of trust.

The very same clients that tell me that they have "the best and brightest" really don't.

They may have a few people that they trust, especially for production facing changes, but for the most part, they are so afraid of the risks that are out there that they become risk adverse in everything.

Do We Develop or Cower in Fear?

Simply getting permission for teams to develop can be a challenge.

For some reason, many of my clients treat their developers just like other information workers that they may employ. For service reps, for example:

- They lock down desktops. Not allowing developers access to tools to allow them to develop is past my comprehension.
- They lock down internet access behind secret proxy servers and very secret firewalls.
- This can lead to developers who have better luck at home, with more open networks, than in the office.
- This can lead to developers who resort to using VPN servers and transparent proxies to allow them to do their work.
- This makes the developers look and act like hackers to security, which further exasperates the problem!

Do You Manage or Develop?

From what I've experienced, the root difference I see between old style siloed and fearful organizations and those that embrace DevOps Culture is whether the organization uses managed services or not.

Managed services means (to me):

- Tickets for everything.
- Tell us your requirements, wait a while, and we'll get back to you with a solution.
- SLAs, Service Portals, Notification and Escalation procedures, etc, etc...

This is the anthesis of DevOps, whether they manage Cloud Based Services or not, because of all the Lean Wastes that come up (waiting, defects, batching/inventory, over-processing, over-production, motion waste, and so on).

What we need is on-demand Infrastructure as Code with a touch of GitOps

- Everything is defined declaratively (such as with Terraform).
- Everything is testable in different environments (the same scripts that produce production infrastructure also produce pre production, UAC, QA, CI, etc.)
- The differences between environments become parameters, such as how many machines are in a cluster, where the endpoints go too, etc.
- When in a low trust environment, as for example an organization transitioning into a DevOps Culture, instead of developers committing changes that will affect production, they create a pull request, and the resulting workflow routes the request to "operations responsible" people. (Side story: It's unreasonable to expect organizations that don't trust developers enough to change their desktop screen saver timeouts to allow those same developers to commit changes directly to production!)
- The "operations responsible" people review the proposed change, merge any conflicts, and commit the change, resulting in a change to production.
- As the organization starts to trust developers to do the right things, the hope is to move from a GitIOps workflow into one with less Lean Waste waiting time, so that changes made by developers can move more directly into production.
- Trust is earned and engendered by a Learning Organization.

Side Benefit: Audit is Fixed!

- One side benefit from this is one the audit concern. In the US, we have a law called Sarbanes-Oxley that was passed back in 2002 to respond to the scandals from Enron, WorldCom, Global Crossing, Tyco, and Authur Andersen.
- One section of that law is section 404, which details the steps that a company must comply with to enable audibility.
- Most companies responded to that section by drafting extensive standardized processes, requiring a ton of sign offs, and so on. Reading some of the defined procedures from companies that I've worked with made me both furious as well as giggle (side story of Everest CSO).
- The result was that when audits occurred, they were met with the same sort of fear that many organizations I've worked with exhibit when releasing code (somehow they forget that releases to production are the normal and desired result of development!)
- Auditors have questions about what changes occurred when, who authorized it, what controls were in place, etc
- The benefit of using a proper CD approach is that we can dispense with the hugely bloated and mostly outdated documents that form the standard operating procedures for these companies and replace them with tools to aid the auditors get whatever information they need from the logs and repositories of the tools used to implement the CD process!
- I've worked with a few chief security officers from companies that were drowning under Sarbanes Oxley and now were some of the most vocal supporters for moving the organization kicking and screaming into the 21st century!

PRODUCTION INFRASTRUCTURE

Where the Action is Now

This is very much the "belly of the beast" when it comes to what I do day to day recently.

I get more requests for "DevOps Coaching" from organizations than any other request.

But most aren't amazing, because they make their decisions on what they fear from "DevOps" companies and tool vendors, who promote the myth that simply buying a few (expensive) tools and implementing them will make them "Agile" and "DevOps" compliant. Probably together and painlessly. They typically don't want to break that the fundamental changes are cultural and feel different than what they do now.

These same clients are simply enamored by the "Agile" process organizations that promote "Instances Scaled-Agile" in their specialty, done with some small changes in process, roles, ceremonies, and certifications. The competition between these Scaled-Agile companies reminds me of a scene from Casablanca (Humphrey Bogart, Peter Lorre, 1942)

But Here's the Rub

The clients I'm familiar with live in fear. Fear of audit. Fear of hacking. Fear of failure.

And lack of trust.

The very same clients that tell me that they have "the best and brightest" really don't.

They may have a few people that they trust, especially for production facing changes, but for the most part, they are so afraid of the risks that are out there that they become risk adverse to everything.

Do We Develop or Cower in Fear?

Simply getting permission for teams to develop can be a challenge.

For some reason, many of my clients treat their developers just like other information workers that they may employ. For service reps, for example:

- They lock down desktops. Not allowing developers access to tools to allow them to develop is past my comprehension.
- They lock down internet access behind secret proxy servers and very secret firewalls.
- This can lead to developers who have better luck at home, with home workstations, than in the office.
- This can lead to developers who resort to using VPN servers and transparent proxies to allow them to do their work.
- This makes the developers look and act like hackers to security, which further exacerbates the problem

Do You Manage or Develop?

From what I've experienced, the truest difference I see between old style siloed and fearful organizations and those that embrace DevOps Culture is whether the organization uses managed services or not.

Managed services means (to me):

- Tickets for everything.
- Tell us your requirements, wait a while, and we'll get back to you with a solution.
- SLAs, Service Portals, Notifications and Escalation procedures, etc, etc...

This is the antithesis of DevOps, whether they manage Cloud Based Services or not, because of all the Lean Wastes that come up (waiting, defects, batching/inventory, over-processing, over-production, and team waste, and so on).

What we need is on-demand Infrastructure as Code with a touch of GitOps

- Everything is defined declaratively (such as with Terraform).
- Everything is testable in different environments (the same scripts that produce production infrastructure also produce the pre-production, UAT, QA, CI, etc.)
- The differences between environments become parameters, such as how many machines are in a cluster, where the endpoints go too, etc.
- When on a Low Trust environment, as for example an organization transitioning into a DevOps Culture, instead of developers committing changes that will affect production, they create a pull request, and the resulting migration begins the request to "operations responsible" people. (Side story: It's unfathomable to expect organizations that don't trust developers enough to change their desktop screen savers tolerants to allow their same developers to commit changes directly to production?)
- The "operations responsible" people review the proposed change, merge any conflicts, and commit the change, resulting in a change to production.
- As the organization learns to trust developers to do the right things, the hope is to move from a GitOps workflow into one with less team waste wasting time, so that changes made by developers can move more directly into production.
- Trust is earned and encouraged by a Learning Organization.

Side Benefit: Audit is Fixed!

- One side benefit from this is one the audit concerns. In the US, we have a law called Sarbanes-Oxley that was passed back in 2002 in response to the scandals from Enron, WorldCom, Global Crossing, Tyco, and Arthur Andersen.
- One section of that law is section 404, which defines the steps that a company must comply with to enable auditability.
- Most companies responded to that section by drafting extensive standardized processes, requiring a ton of signoffs, and so on. Reading some of the defined procedures from companies that I've worked with made me both furious as well as giggle (side story of everest CSO).
- The result was that when audits occurred, they were met with the same sorts of fear that many organizations I've worked with exhibit when releasing code (somehow they forget that releases to production are the normal and desired result of development?)
- Auditors have questions about what changes occurred where, who authorized it, what controls were in place, etc
- The benefit of using a proper CD approach is that we can dispense with the hugely relayed and mostly outdated agreements that push the standard operating procedures for siloe companies and replace them with tools to aid the auditors get whatever information they need from the logs and repositories of the tools used to implement the CD process?
- I've worked with a few chief security officers from companies that were drowning under Sarbanes Oxley and now were some of the most vocal supporters for moving the organization kicking and screaming into the 21st century!

Who Am I?

On Requirements Gathering

What About Planning?

Development environments

Development Infrastructure

On Coding Practices

Let's Talk About Testing

Production Infrastructure

Developing a Learning Organization

The Tale for the Misbegotten

So Long,
and Thanks for
All the Fish

Questions,
Answers,
Debate,
Dispute

Developing a Learning Organization

Finding A New Role for the HiPPOs

FACTS MATTER

How to develop a learning organization. That's probably the thorniest issue of anything discussed thus far, because it goes to the very heart of the organization's culture.

It's hard because we've been taught that leaders, who usually make more money than the others in the room, are supposed to have the answers to all the questions that the organization has.

It's that HiPPO (Highest Paid Person's Opinion) decision making that is so dangerous.

Evidence based and data driven decision making is the right way to go — otherwise you are just exercising old style command and control, and making decisions at the wrong level.

How HiPPOs Create Fear

So, while there's plenty of work for a person like myself to coach executives on their personal style, it's the bleed of current HiPPO decisions into the organization that affects us all in the here and now.

Yes. We need to turn our managers into servant leaders, but we can't wait until that happens before we start to make a change for the better!

That's because in an organization where there's fear, most people will respond by internalizing that a good survival strategy is to stay quiet, listen, and do what they are told to do.

As if that will help them survive the next round of layoffs!

Why Organizations Have to Fail to Survive

The world level view of all of this is very much like the world level view of evolution.

Innovations form the normal organism itself, are good and lead to dominance because of advantage, or they cause the organism (and all future offspring) to die.

Unless all we want to do is react to customer demand and try to compete on an ever shrinking profit by doing what everyone else is doing, but doing it cheaper, we have to find that advantage.

Advantage always starts with a capability that others don't have.

As that advantage becomes a commodity, we have to find another advantage.

There's no way (that works) for managers to tell their reports, "Give me two innovations that will propel us to the next level by Tuesday!"

Creativity and innovation just don't work on command.

Start Learning by Starting to Trust

So, how do you get your organization to become a learning organization? (Isn't every learning organization already, since all they ever seem to be doing is being and knowing and working toward self improvement and other things they shouldn't have dreamed of for themselves once, and it's not so far away from you?)

I have a few tips that I can share with you:

- You could do more and trust less
- Don't leave the legacy of past exemplars with a magic wave...
- Your teams tackle a living test that is because it...
- Get people the opportunities to learn...
- Celebrate failure...
- Find ways for people to connect...

Engendering Innovation

We're not the first people that have had to deal with engendering innovation at an organizational level.

I draw a lot of inspiration from Mervin Kelly.

The Choice is Yours

So my question to you is how can we be more like Mervin and less like Frederick Taylor, who's legacy in scientific management is mostly the stopwatch and time motion studies. Taylor was once denounced by Lenin in 1913 as "the scientific system of sweating more work from laborers".

What do we want the legacy for our organizations to be?

More like the guys who are great efficient extensions of their manufacturing machines, or the guys who invented the transistor and helped establish modern telecommunication and computing for the ages?

Finding A New Role for the HiPPOs

FACTS MATTER

How to develop a Learning Organization. That's probably the thorniest issue of anything discussed thus far, because it goes to the very heart of the organization's culture.

It's hard because we've been taught that leaders, who usually make more money than the others in the room, are supposed to have the answers to all the questions that the organization has.

It's that HiPPO (Highest Paid Person's Opinion) decision making that is so dangerous.

Evidence based and data driven decision making is the right way to go – otherwise you are just exercising old style command and control and making decisions at the wrong level.

How HiPPOs Create Fear

So, while there's plenty of work for a person like myself to coach executives on their personal style, it's the bleed of current HiPPO decisions into the organization that affects us all in the here and now.

Yes. We need to turn our managers into servant leaders, but we can't wait until that happens before we start to make a change for the better!

That's because in an organization where there's fear, most people will respond by internalizing that a good survival strategy is to stay quiet, listen, and do what they are told to do.

As if that will help them survive the next round of layoffs!

Why Organizations Have to Fail to Survive

The world level view of all of this is very much like the world level view of evolution.

Mutations from the normal organism either are good and lead to dominance because of advantage, or they cause the organism (and all future offspring) to die.

Unless all we want to do is react to customer demand and try to compete on an ever shrinking profit by doing what everyone else is doing, but doing it cheaper, we have to find that advantage.

Advantage always starts with a capability that others don't have.

As that advantage becomes a commodity, we have to find another advantage.

There's no way (that works) for managers to tell their reports, "Give me two innovations that will propel us to the next level by Tuesday!"

Creativity and innovation just don't work on command.

Start Learning by Starting to Trust

The Iceberg Illusion

Success is an iceberg

SUCCESS!

WHAT PEOPLE SEE

WHAT PEOPLE DON'T SEE

Persistence

Failure

FAIL

Sacrifice

THINGS I HAVE TO GIVE UP

Disappointment

Dedication

GOAL
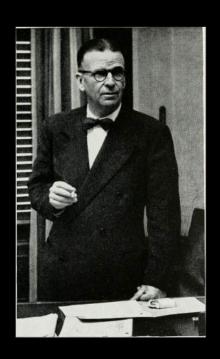
Hard work

Good habits

@sylviaduckworth

So, how do you get your organization to become a learning organization? [Side story: signage doesn't do it. Signs all over Bayer extolling science and knowledge, with exultations to experiment and learn. But mess up and don't make the dates or the numbers once, and it's off to the gallows for you!]

I'd have to say that it starts with trust.

- Hire excellent people and trust them.
- Don't hire just because of past experience with a particular tool or framework. That sort of stuff changes all the time. [Side story: I started programming computers 51 years ago, and I'm proud to say that I'm still learning new and better ways to program!]
- Hire people that can learn. Test that as part of the interview process. Forget about the resume as much as possible.
- Give people the opportunities to learn. That means books, courses, conferences, budget for buying things, and so on.
- Celebrate failure. I almost mean literally! We tend to extol those who figure something out, but try to sweep failures under the rug. Try to find that nugget of learning, even if what we learn is "that was a dumb idea." Education costs money.
- Find ways for people to connect and learn from one another. Especially from people they are not used to working with. As in those Hackathons I spoke about earlier.

Engendering Innovation

We're not the first people that have had to deal with engendering innovation at an organizational level.

I draw a lot of inspiration from Mervin Kelly.

Life at Bell Labs

Mervin trained as a physicist at the University of Chicago. His PhD advisor was Robert Millikan (the guy who devised the oil drop experiment to measure the elemental charge on the electron).

By 1925, Mervin went to Bell Labs, and rose from researcher to chairman of the board.

His fundamental belief was that an "institute of creative technology" needed a "critical mass" of talented people to foster a busy exchange of ideas.

Physical Proximity Was Key to Mervin

But innovation required much more than that. Mervin was convinced that physical proximity was everything. Phone calls alone wouldn't do.

So, quite intentionally, Bell Labs mixed theorists and doers under one roof.

When Bell Labs was developing the transistor, he made sure that physicists, metallurgists, and electrical engineers were all side by side so theory, experimentation and fabrication plans could come together.

Like a rambunctious researcher, he sought a workshop, and sometimes a tension, between scientific disciplines, between researchers and developers, and between scientists and groups.

Like a Magnet Rolling Past Iron Filings

One surprising element of his approach was architectural.

He personally helped design the building in Murray Hill, N.J. which opened in 1941, where everyone would interact with one another.

Some of the hallways in the building were designed to be so long that to look down their length was to see the end and disappear at a vanishing point.

Traveling the hall's length without encountering a number of acquaintances, problems, questions and ideas was almost impossible.

A physicist on his way to lunch in the cafeteria was like a magnet rolling past iron filings.

Life at Bell Labs

Mervin trained as a physicist at the University of Chicago. His PhD advisor was Robert Millikan (the guy who devised the oil drop experiment to measure the elemental charge on the electron).

By 1925, Mervin went to Bell Labs, and rose from researcher to chairman of the board.

His fundamental belief was that an "institute of creative technology" needed a "critical mass" of talented people to foster a busy exchange of ideas.

Physical Proximity Was Key to Mervin

But innovation required much more than that. Mervin was convinced that physical proximity was everything; phone calls alone wouldn't do.

So, quite intentionally, Bell Labs housed thinkers and doers under one roof.

When Bell Labs was developing the transistor, he made sure that physicists, metallurgists and electrical engineers were all side by side so theory, experimentation and manufacturing could come together.

Like a concert hall conductor, he sought a harmony, and sometimes a tension, between scientific disciplines; between researchers and developers; and between soloists and groups.

Like a Magnet Rolling Past Iron Filings

One surprising element of his approach was architectural.

He personally helped design the building in Murray Hill, N.J. which opened in 1941, where everyone would interact with one another.

Some of the hallways in the building were designed to be so long that to look down their length was to see the end disappear at a vanishing point.

Traveling the hall's length without encountering a number of acquaintances, problems, diversions and ideas was almost impossible.

A physicist on his way to lunch in the cafeteria was like a magnet rolling past iron filings.

Engendering Innovation

We're not the first people that have had to deal with engendering innovation at an organizational level.

I draw a lot of inspiration from Mervin Kelly.

Life at Bell Labs

Mervin trained as a physicist at the University of Chicago. His PhD advisor was Robert Millikan (the guy who devised the oil drop experiment to measure the elemental charge on the electron).

By 1925, Mervin went to Bell Labs, and rose from researcher to chairman of the board.

His fundamental belief was that an "institute of creative technology" needed a "critical mass" of talented people to foster a busy exchange of ideas.

Physical Proximity Was Key to Mervin

But innovation required much more than that. Mervin was convinced that physical proximity was everything; phone calls alone wouldn't do.

So, quite intentionally, Bell Labs merged theorists and doers under one roof.

When Bell Labs was developing the transistor, he made sure that physicists, metallurgists, and electrical engineers were all side by side so nearly simultaneous and haphazard phone could come together.

Like a genius roaming researcher, he sought a mandate, and sometimes a tension, between scientific disciplines, between researchers and developers, and between scientists and groups.

Like a Magnet Rolling Past Iron Filings

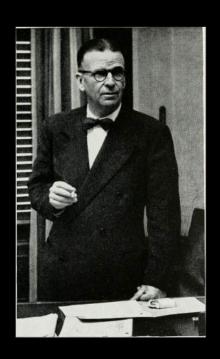One surprising element of his approach was architectural.

He personally helped design the building in Murray Hill, N.J. which opened in 1941, where everyone would interact with one another.

Some of the hallways in the building were designed to be so long that to look down their length was to see far and disappear at a vanishing point.

Traveling the hall's length without encountering a number of acquaintances, problems, distractions and ideas was almost impossible.

A physicist on his way to lunch in the cafeteria was like a magnet rolling past iron filings.

The Choice is Yours

So my question to you is how can we be more like Mervin and less like Frederick Taylor, who's legacy in scientific management is mostly the stopwatch and time motion studies. Taylor was once denounced by Lenin in 1913 as "the scientific system of sweating more work from laborers".

What do we want the legacy for our organizations to be?

More like the guys who are great efficient extensions of their manufacturing machines, or the guys who invented the transistor and helped establish modern telecommunication and computing for the ages?

Developing a Learning Organization

Finding A New Role for the HiPPOs

FACTS MATTER

How to develop a learning organization. That's probably the thorniest issue of anything discussed thus far, because it goes to the very heart of the organization's culture.

It's hard because we've been taught that leaders, who usually make more money than the others in the room, are supposed to have the answers to all the questions that the organization has.

It's that HiPPO (Highest Paid Person's Opinion) decision making that is so dangerous.

Evidence based and data driven decision making is the right way to go - otherwise you are just exercising old style command and control, and making decisions at the wrong level.

How HiPPOs Create Fear

So, while there's plenty of work for a person like myself to coach executives on their personal style, it's the bleed of current HiPPO decisions into the organization that affects us all in the here and now.

Yes. We need to turn our managers into servant leaders, but we can't wait until that happens before we start to make a change for the better!

That's because in an organization where there's fear, most people will respond by internalizing that a good survival strategy is to stay quiet, listen, and do what they are told to do.

As if that will help them survive the next round of layoffs!

Why Organizations Have to Fail to Survive

The world level view of all of this is very much like the world level view of evolution.

Innovations form the normal organism itself are good and lead to dominance because of advantage, or they cause the organism (and all future offspring) to die.

Unless we all want to do it react to customer demand and try to compete on an even smaller profit by doing what everyone else is doing, but doing it cheaper, we have to find that advantage.

Advantage always starts with a capability that others don't have.

As that advantage becomes a commodity, we have to find another advantage.

There's no way (that works) for managers to tell their reports, "Give me two innovations that will propel us to the next level by Tuesday!"

Creativity and innovation just don't work on command.

Start Learning by Starting to Trust

So, how do you get your organization to become a learning organization? (Such short thorny advice ain't. Scaling are ever super upraling science and knowledge, within science to experiment and experiments that don't want this ahead of for the more sort, and it's off to be success for you?)

I haven't it up that it starts with trust.

- You feels like you're and trust them
- Don't fear detriment of mastery/learn within a fault idea for an framework. That way so they changes as the same. I used often I started programming on computers 35 years ago, and I'm proud to say that I'm still learning now and after now so fascinating!
- Put people into learning. Test may be as part of the natural process. Trust's about the feeling as when in progress.
- Give people the opportunities to learn. That means books, classes, conferences, budget for buying things, and such.
- Celebrate failure. I always learn literally! We tend to feel bad that people failure something out, but try to avoid failures from the it. Try to find that mindset of learning, even if what we learn it "that was a bummer." Education is learning.
- Find ways for people to connect and along from one another. Especially the people they are not like to working with. It's so delightful to learn and enjoy it.

Engendering Innovation

We're not the first people that have had to deal with engendering innovation at an organizational level.

I draw a lot of inspiration from Mervin Kelly.

The Choice is Yours

So my question to you is how can we be more like Mervin and less like Frederick Taylor, who's legacy in scientific management is mostly the stopwatch and time motion studies. Taylor was once denounced by Lenin in 1913 as "the scientific system of sweating more work from laborers".

What do we want the legacy for our organizations to be?

More like the guys who are great efficient extensions of their manufacturing machines, or the guys who invented the transistor and helped establish modern telecommunication and computing for the ages?

Who Am I?

On Requirements Gathering

What About Planning?

Development environments

Development Infrastructure

On Coding Practices

Let's Talk About Testing

Production Infrastructure

Developing a Learning Organization

The Tale for the Misbegotten

The Tale for the Misbegotten

What's With the Reference in the Presetation's Name?

As I reflect on my journey as an Agile coach and the twists and turns over the years as I've worked this role, it sometimes feels that I'm a character in the classic Eugene O'Neill play called "A Moon for the Misbegotten".

The play centers around an cynical alcoholic who lives in Connecticut and has long standing mother issues, sibling issues, and neighbor issues.

Nostalgic for An Old Friend

Well, I don't really drink, but sometimes I feel quite cynical, sort of in the Andy Rooney way (a commenter on "60 Minutes", famous for his "A Few Minutes with Andy Rooney" segments from the late 1970s to the late 2000s.)

To get to know Andy, take a look at "Andy Bottles Eau De Rooney", where he satirically rips apart the trend from using tap water to bottled water in the US.

As I age, I get a sense that I've become the same sort of crabby old man that Andy was, except instead of bemoaning about how writing (and all sorts of other everyday things) have changed for the worse in the modern world, I whine about how software development has "just gone to the dogs."

I feel that I share a lot with Andy, and miss his commentary a lot.

Fighting the Good Fight

I'm sure that if Andy ever saw the industrialized and over-mechanized process for software development and deployment that's become the standard these days, I think the curmudgeon in its both would say something like, "Did you ever notice that things used to be different back in the day, when we did better, were happier, and didn't have all those Software Development tools, processes, certifications, and the like?"

But I feel passionately about this, and that's why I've come so far here to share this passion with you.

Because we'll lose something important if we fail to remember what's really important in our field and fight to keep the flame alive.

At it's essence, I feel that DevOps is the realization that industrialized and mechanized Software Development tools and processes is nothing more than a scam that makes for a good business for the vendors of the tools and processes that they hawk.
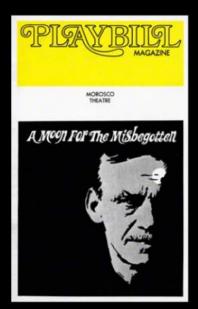
We must stand strong and fight the good fight!
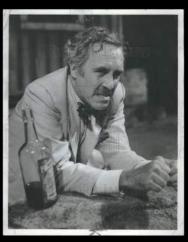
So Long,
and Thanks for
All the Fish

Questions.
Answers.
Debate.
Depart.

What's With the Reference in the Presetation's Name?

As I reflect on my journey as an Agile coach and the twists and turns over the years as I've worked this role, it sometimes feels that I'm a character in the classic Eugene O'Neill play called "A Moon for the Misbegotten".

The play centers around an cynical alcoholic who lives in Connecticut and has long standing mother issues, sibling issues, and neighbor issues.

Nostalgic for An Old Friend

Well, I don't really drink, but sometimes I feel quite cynical, sort of in the Andy Rooney way (a commenter on "60 Minutes", famous for his "A Few Minutes With Andy Rooney" segment from the late 1970s to the late 2000s.)

To get to know Andy, take a look at "Andy Bottles Eau De Rooney", where he satirically rips apart the trend from using tap water to bottled water in the US.

As I age, I get a sense that I've become the same sort of crabby old man that Andy was, except instead of bemoaning about how writing (and all sorts of other everyday things) have changed for the worse in the modern world, I whine about how software development has "just gone to the dogs."

I feel that I share a lot with Andy, and miss his commentary a lot.

Nostalgic for An Old Friend

Well, I don't really drink, but sometimes I feel quite cynical, sort of in the Andy Rooney way (a commenter on "60 Minutes", famous for his "A Few Minutes With Andy Rooney" segment from the late 1970s to the late 2000s.)

To get to know Andy, take a look at "Andy Bottles Eau De Rooney", where he satirically rips apart the trend from using tap water to bottled water in the US.

As I age, I get a sense that I've become the same sort of crabby old man that Andy was, except instead of bemoaning about how writing (and all sorts of other everyday things) have changed for the worse in the modern world, I whine about how software development has "just gone to the dogs."

I feel that I share a lot with Andy, and miss his commentary a lot.

Fighting the Good Fight

I'm sure that if Andy ever saw the industrialized and over mechanized process for software development and deployment that's become the standard these days, I think the curmudgeon in us both would say something like, "Did you ever notice that things used to be different back in the day, when we did better, were happier, and didn't have all those Software Development tools, processes, certifications, and the like?"

But I feel passionately about this, and that's why I've come so far here to share this passion with you.

Because we'll lose something important if we fail to remember what's really important in our field and fight to keep the flame alive.

At it's essence, I feel that DevOps is the realization that industrialized and mechanized Software Development tools and processes is nothing more than a scam that makes for a good business for the vendors of the tools and processes that they hawk.

We must stand strong and fight the good fight!

So Long, and Thanks for All the Fish

Questions.
Answers.
Debate.
Depart.

The Tale for the Misbegotten

What's With the Reference in the Presetation's Name?

As I reflect on my journey as an Agile coach and the twists and turns over the years as I've worked this role, it sometimes feels that I'm a character in the classic Eugene O'Neill play called "A Moon for the Misbegotten".

The play centers around an cynical alcoholic who lives in Connecticut and has long standing mother issues, sibling issues, and neighbor issues.

Nostalgic for An Old Friend

Well, I don't really drink, but sometimes I feel quite cynical, sort of in the Andy Rooney way (a commenter on "60 Minutes", famous for his "A Few Minutes with Andy Rooney" segments from the late 1970s to the late 2000s.)

To get to know Andy, take a look at "Andy Bottles Eau De Rooney", where he satirically rips apart the trend from using tap water to bottled water in the US.

As I age, I get a sense that I've become the same sort of crabby old man that Andy was, except instead of bemoaning about how writing (and all sorts of other everyday things) have changed for the worse in the modern world, I whine about how software development has "just gone to the dogs."

I feel that I share a lot with Andy, and miss his commentary a lot.

Fighting the Good Fight

I'm sure that if Andy ever saw the industrialized and over mechanized process for software development and deployment that's become the standard these days, I think the curmudgeon in us both would say something like, "Did you ever notice that things used to be different back in the day, when we did better, were happier, and didn't have all those Software Development tools, processes, certifications, and the like?"

But I feel passionately about this, and that's why I've come so far here to share this passion with you.

Because we'll lose something important if we fail to remember what's really important in our field and fight to keep the flame alive.

At it's essence, I feel that DevOps is the realization that industrialized and mechanized Software Development tools and processes is nothing more than a scam that makes for a good business for the vendors of the tools and processes that they hawk.

We must stand strong and fight the good fight!

So Long, and Thanks for All the Fish

Questions.
Answers.
Debate.
Depart.

Who Am I?

On Requirements Gathering

What About Planning?

Development environments

Development Infrastructure

On Coding Practices

Let's Talk About Testing

Production Infrastructure

Developing a Learning Organization

The Tale for the Misbegotten

Pivoting a Company Towards GitOps and Continuous Delivery

A Technical Practices Coach's Tale for the Misbegotten

DEVOPS PRO EUROPE

Howard Deiner