TECHNICAL DEBT TRAP

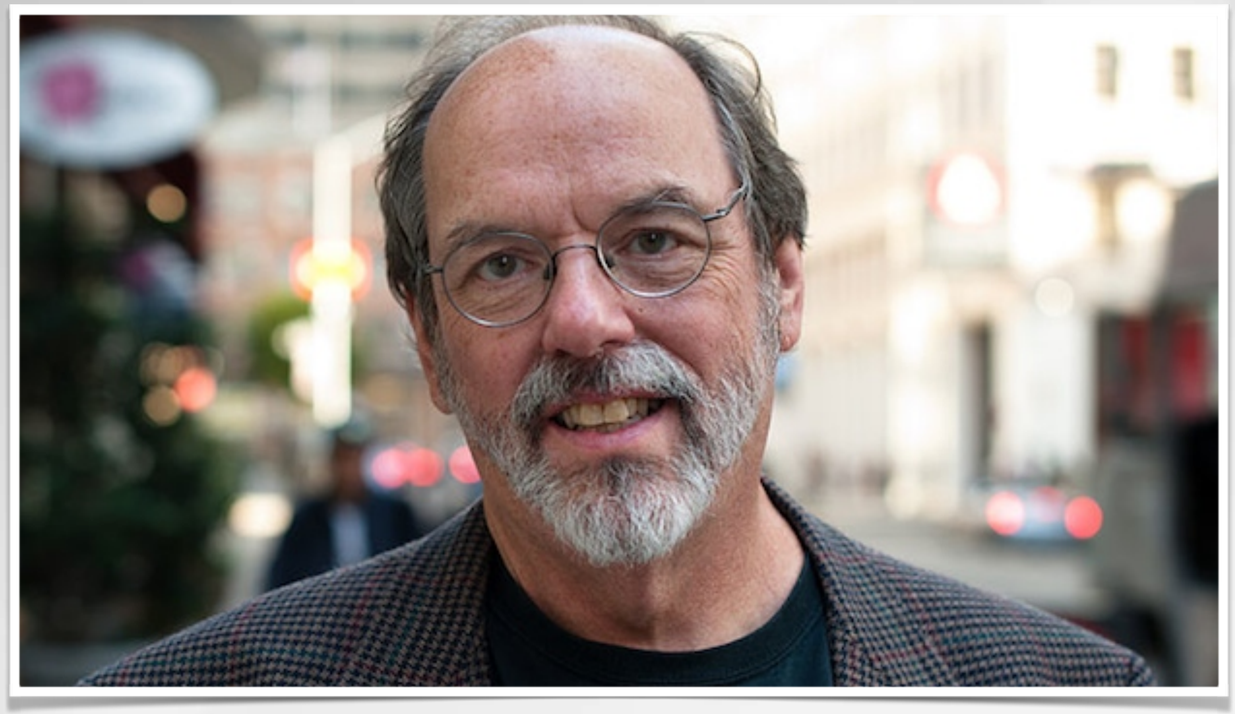
CTO2

TALENT + TECHNOLOGY

#etka17 / #TechnicalDebt / @DocOnDev / @WeAreCTO2

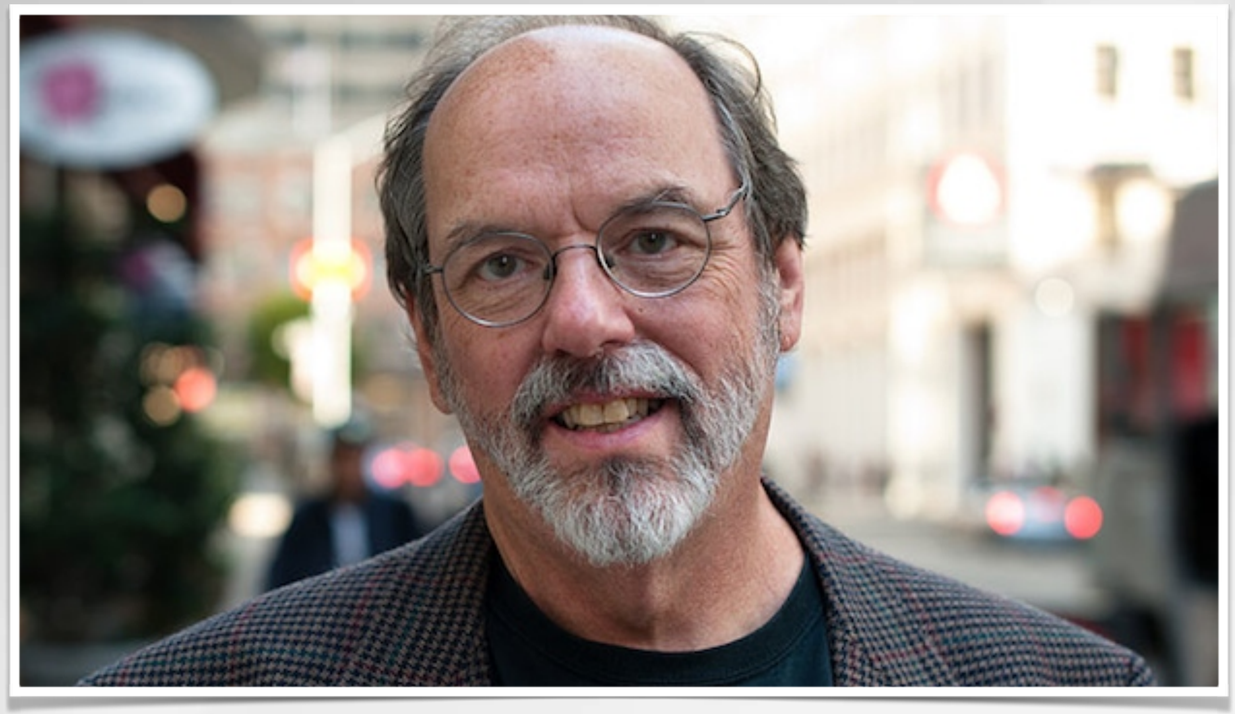
DOC NORTON
CO-FOUNDER + CEO
DOC@WEARECTO2.COM

WHAT IS TECHNICAL DEBT?



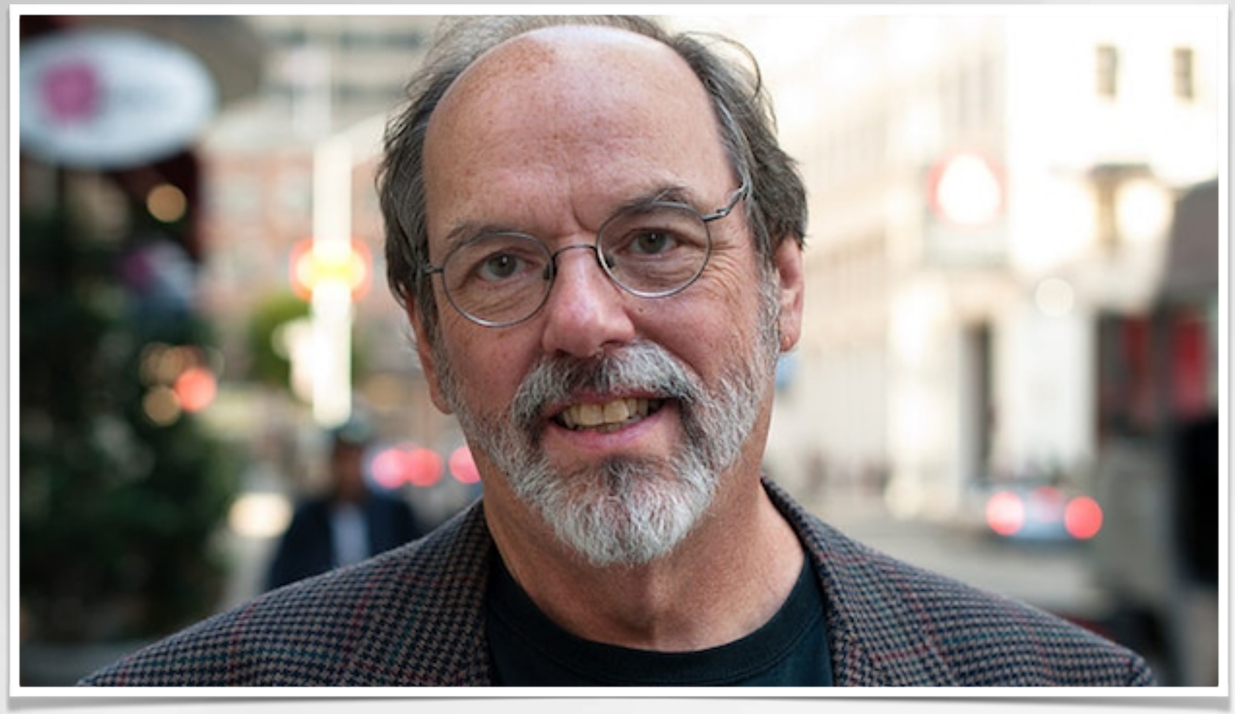
“Shipping first time code is like going into debt.”

–WARD CUNNINGHAM :: OOPSLA ‘92



"Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite."

–WARD CUNNINGHAM :: OOPSLA '92

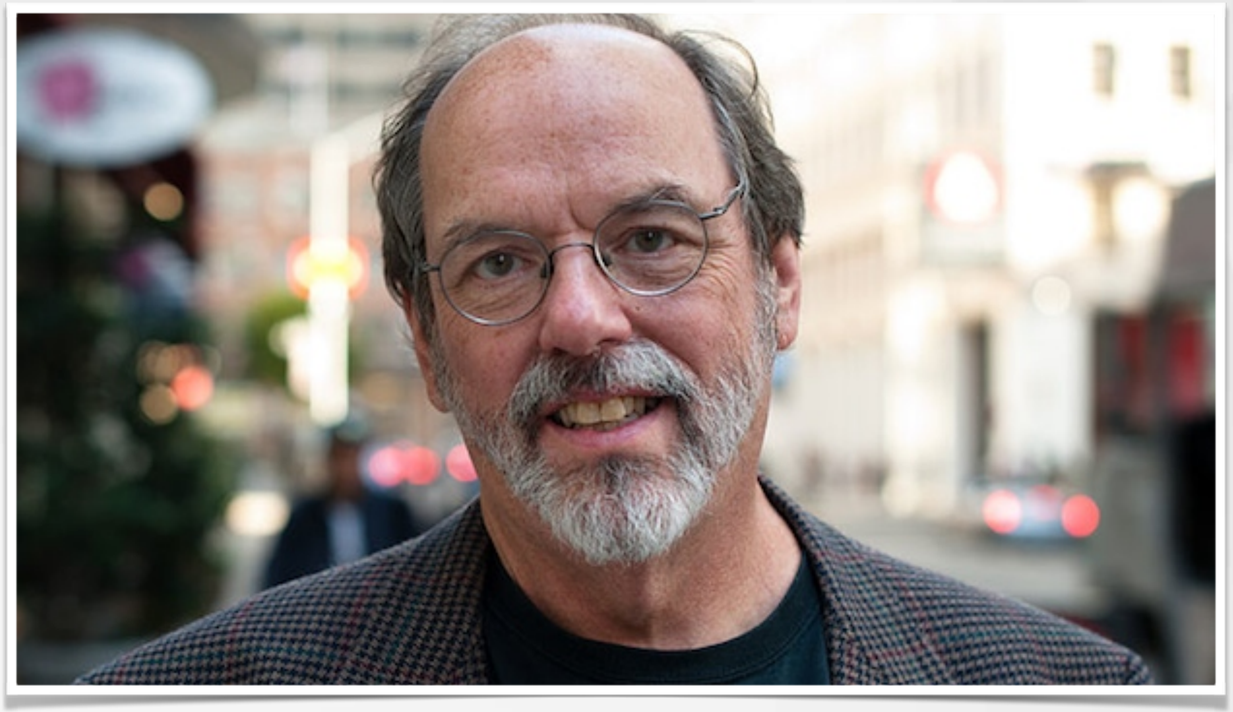


"Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite.

The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt."

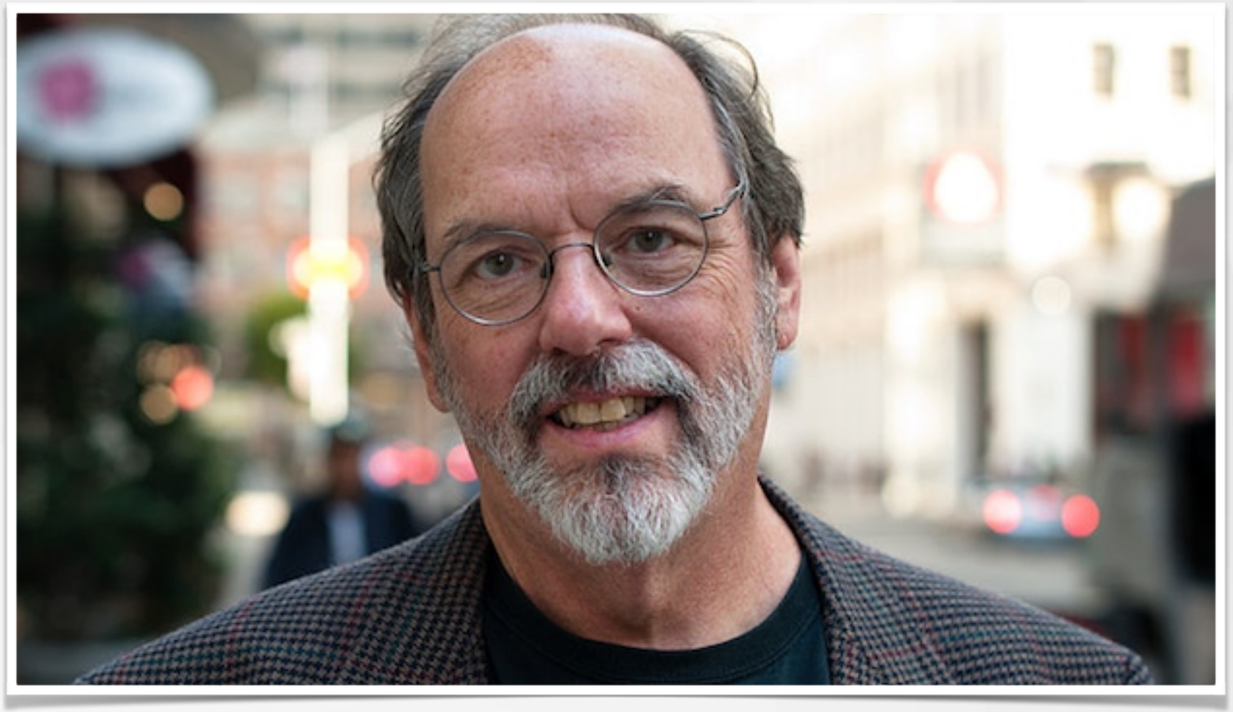
–WARD CUNNINGHAM :: OOPSLA '92

“The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt.”



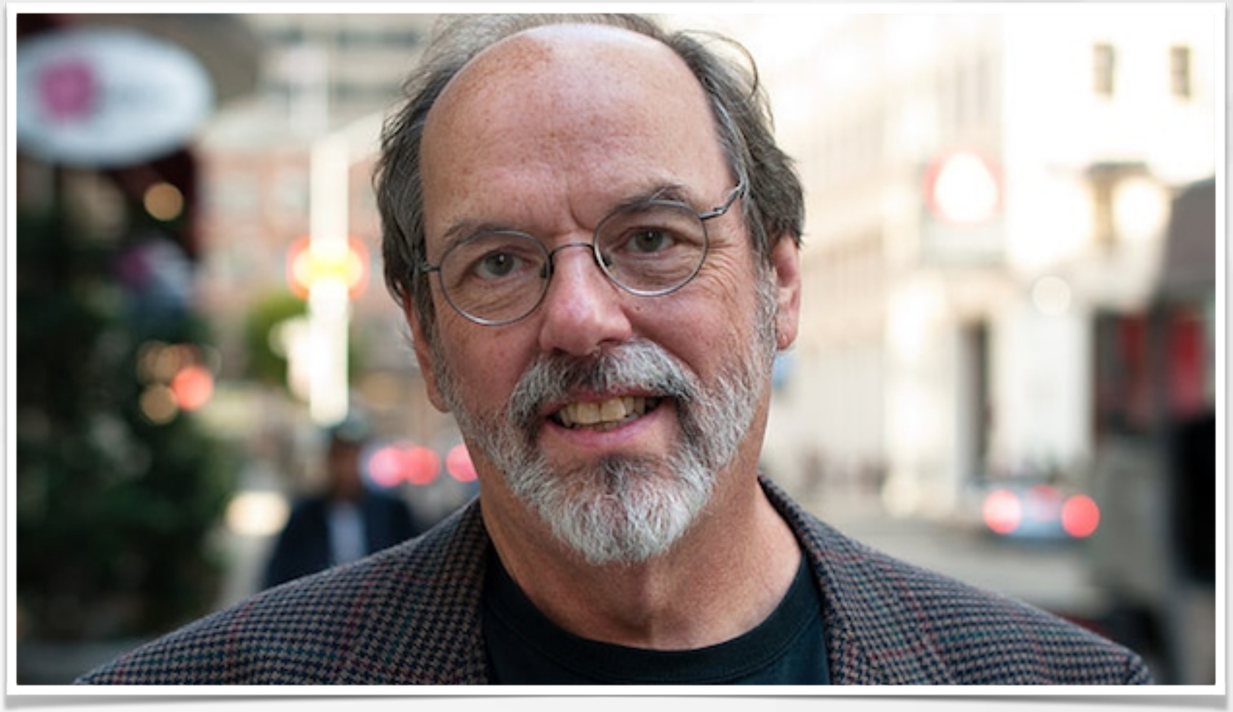
–WARD CUNNINGHAM :: OOPSLA ‘92

“The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt.”



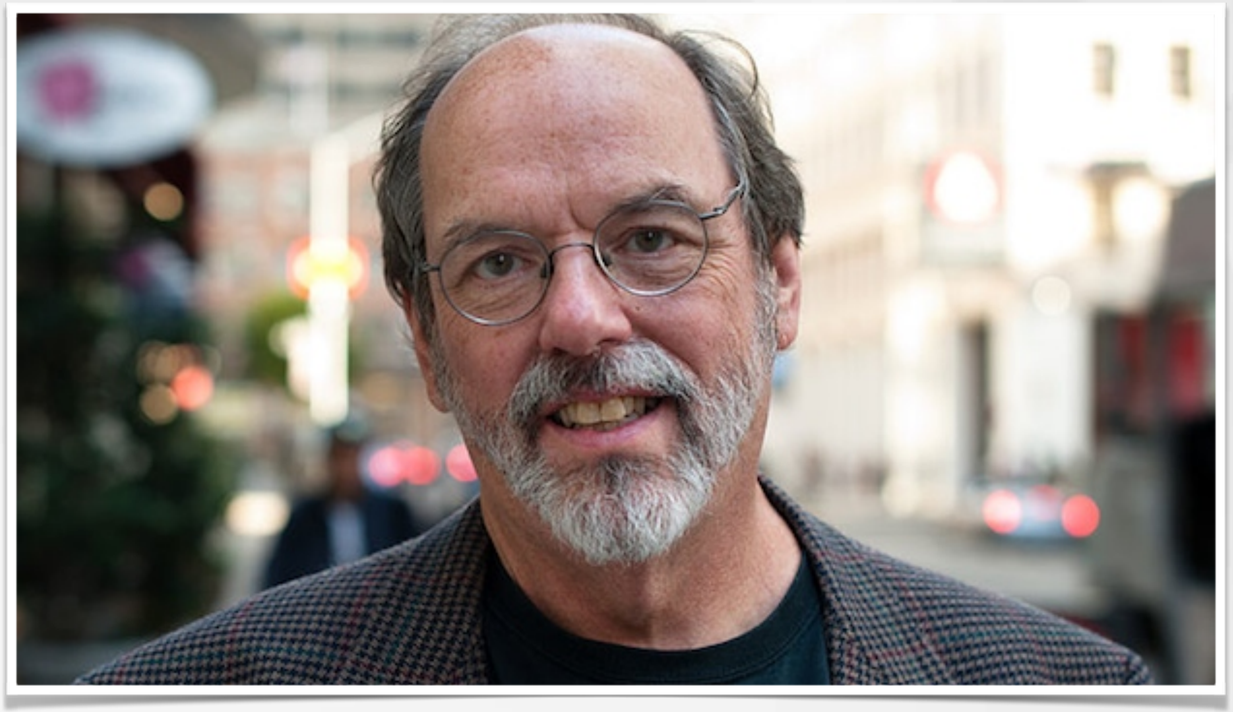
–WARD CUNNINGHAM :: OOPSLA ‘92

“The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt.”



–WARD CUNNINGHAM :: OOPSLA ‘92

“The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt.”



–WARD CUNNINGHAM :: OOPSLA ‘92

TECHNICAL DEBT IS

TECHNICAL DEBT IS GOOD


TECHNICAL DEBT IS GOOD

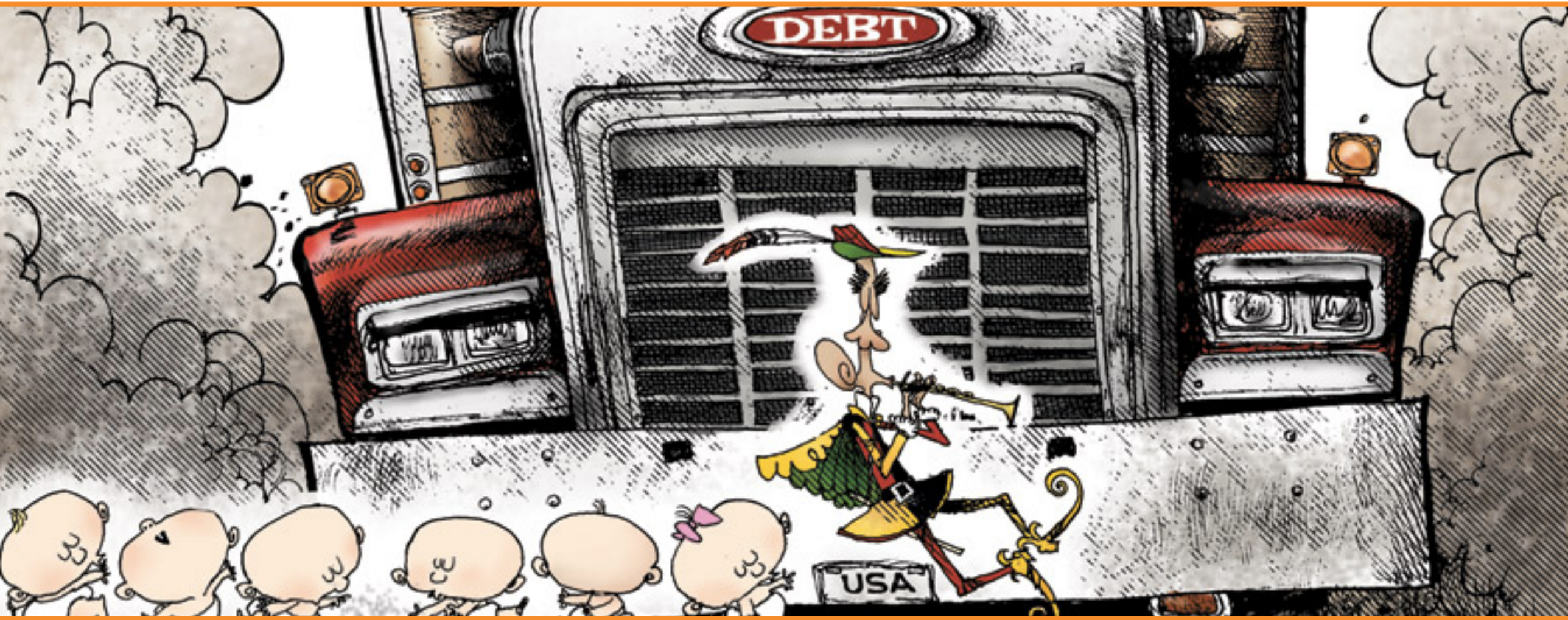
TECHNICAL DEBT IS GOOD

TECHNICAL DEBT IS A STRATEGIC DECISION

- Allow for Rapid Delivery
- To Elicit Quick Feedback
- And Correct Design

TECHNICAL DEBT IS AN INDICATION OF LEARNING

- Now know what you need
- Implementation doesn't match



TECHNICAL DEBT IS A METAPHOR

CTO2
TALENT + TECHNOLOGY

Can't keep running at this pace

BUILDING ON A WEAK FOUNDATION

PUTS PRESSURE ON OUR DESIGN

IT'S RAINING MEN (HALLELUJAH)

METAPHORS ROCK

we reason by analogy

CTO2 TALENT + TECHNOLOGY

METAPHORPHOSIS

when metaphors go wrong

"CUT A LOT OF CORNERS"

- James Shore

"QUICK AND DIRTY"

- Martin Fowler

"JUST HACK IT IN"

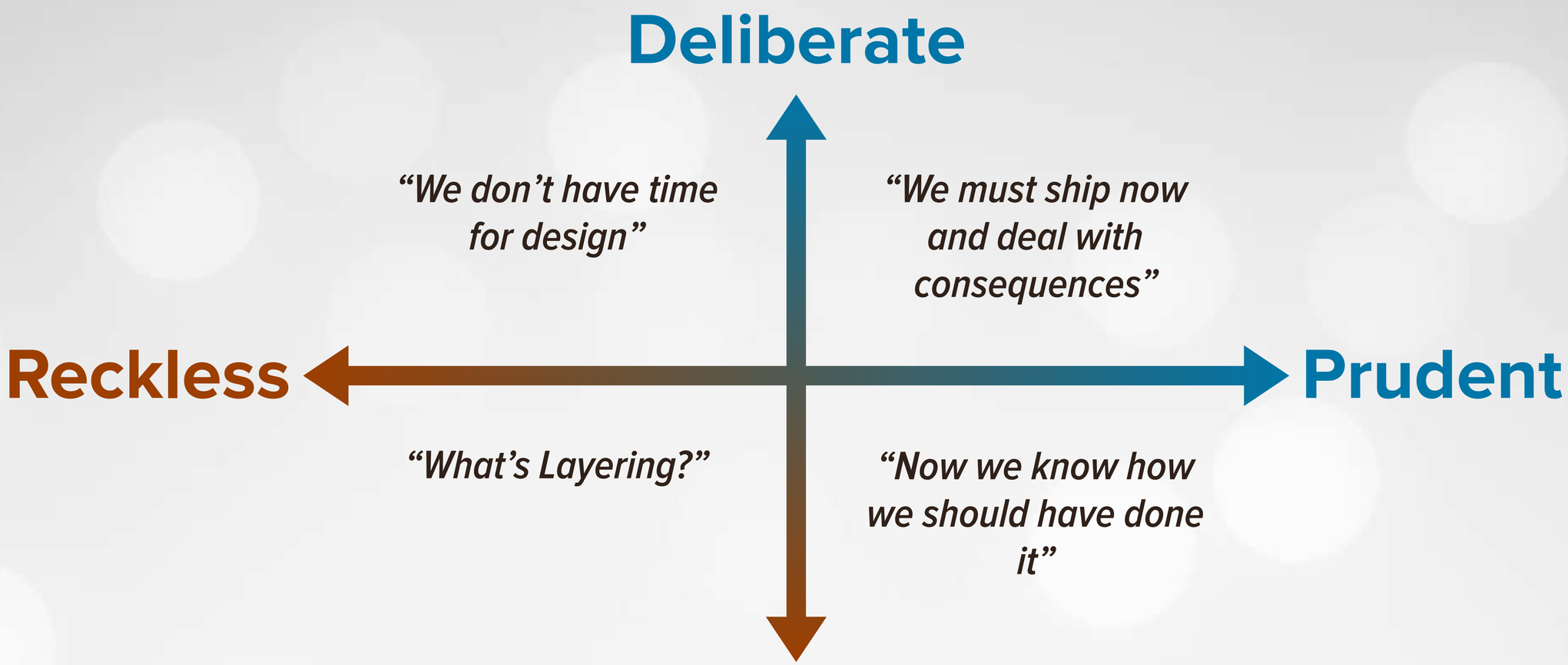
- Steve McConnell

"SLOPPY"

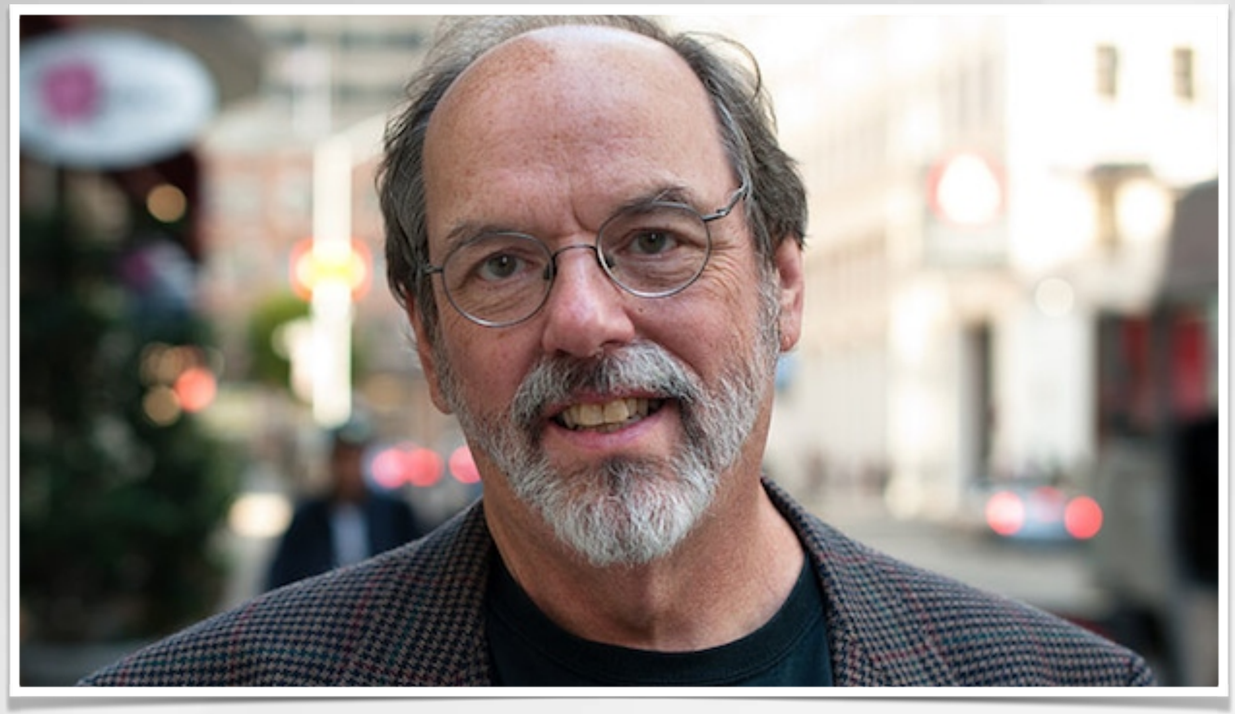
- David Laribee

METAPHORPHOSIS

when metaphors go wrong

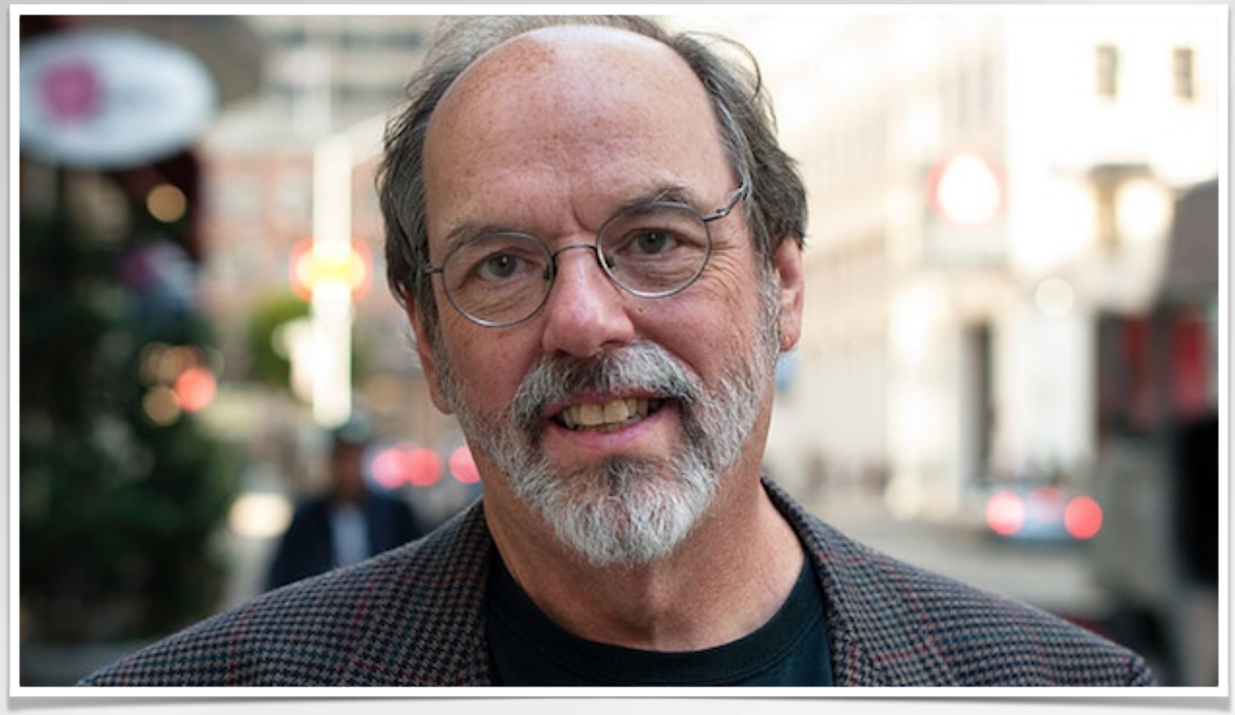


TECHNICAL DEBT QUADRANT



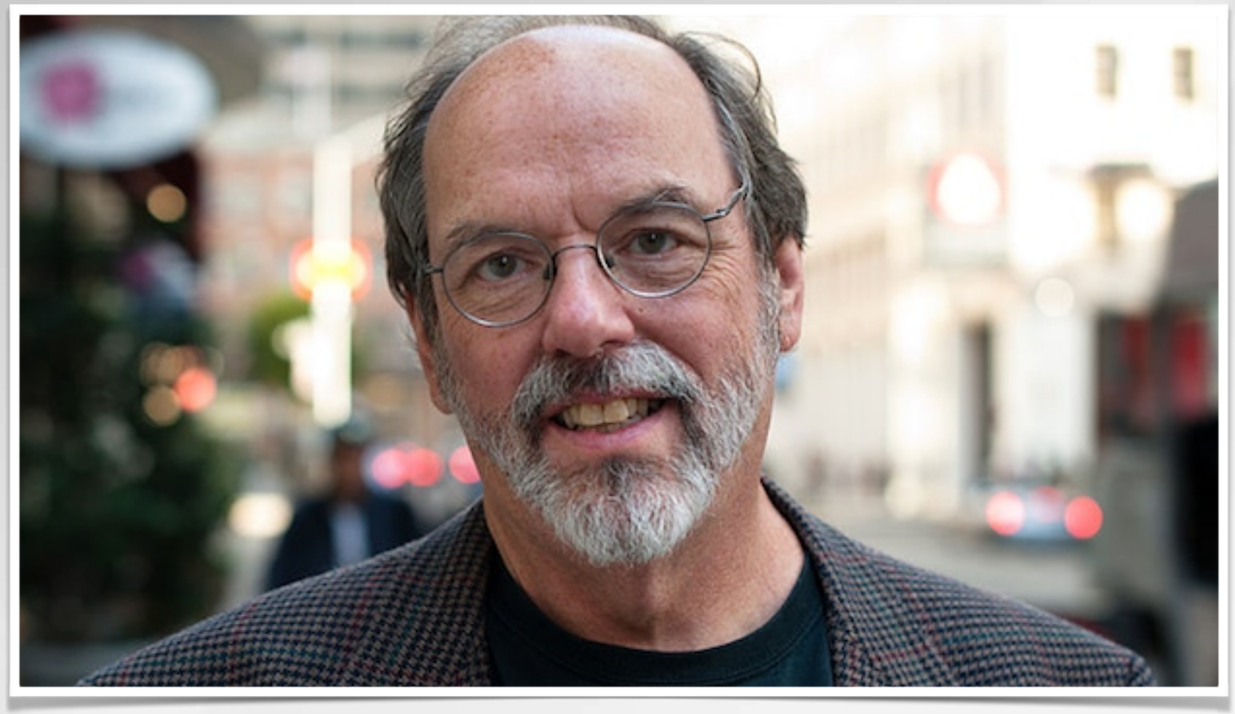
“[Many] have explained the debt metaphor and confused it with the idea that you could write code poorly with the intention of doing a good job later.”

–WARD CUNNINGHAM :: YOUTUBE ‘09



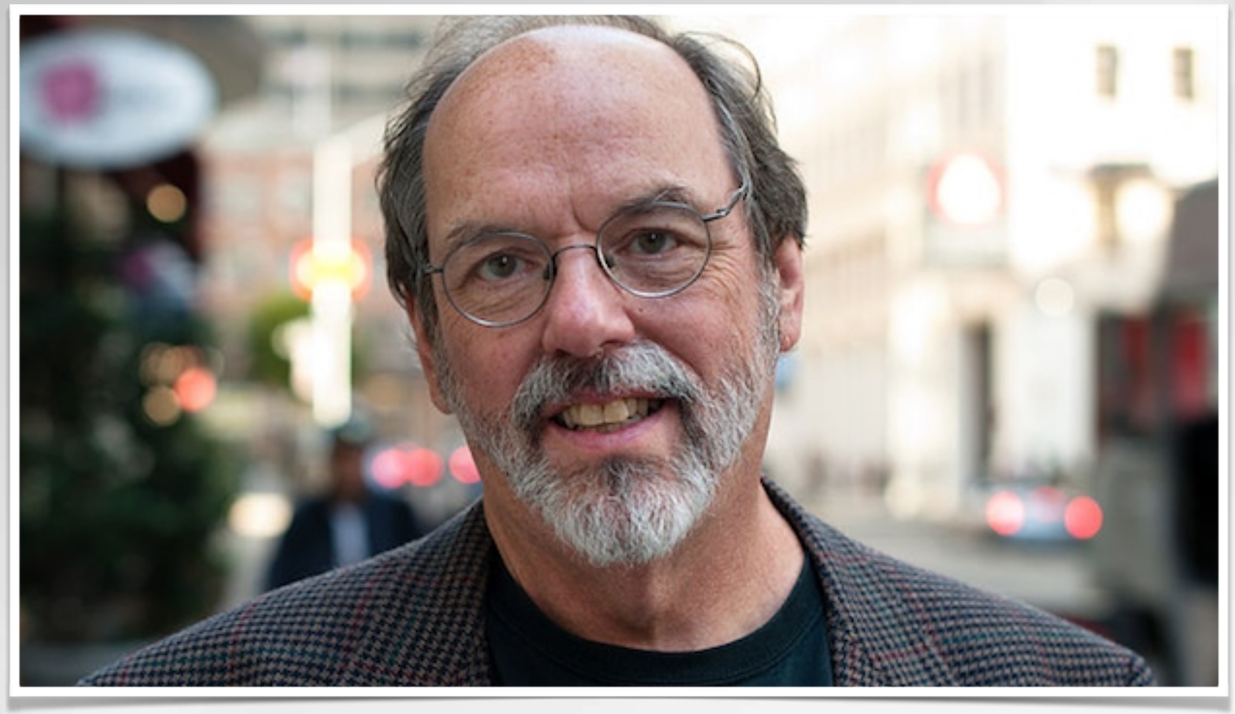
"...confused the debt metaphor with the idea that you could write code poorly..."

–WARD CUNNINGHAM :: YOUTUBE '09



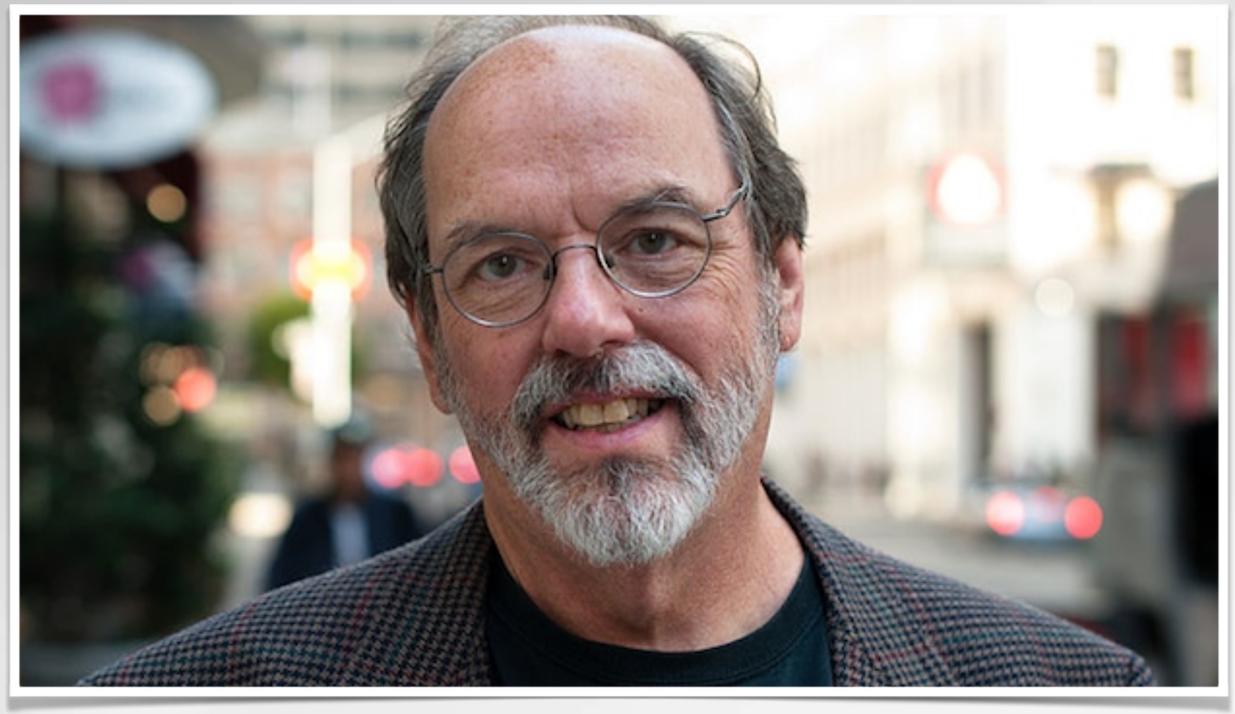
“The ability to pay back debt [...] depends upon you writing code that is clean enough to be able to refactor as you come to understand your problem.”

–WARD CUNNINGHAM :: YOUTUBE ‘09



“The ability to pay back debt [...] depends upon you writing code that is ***clean enough to be able to refactor*** *as you come to understand your problem.”*

–WARD CUNNINGHAM :: YOUTUBE ‘09



“Dirty code is to technical debt as the pawn broker is to financial debt.”

“Don’t think you are ever going to get your code back.”

–WARD CUNNINGHAM :: TWITTER ‘09

TECHNICAL DEBT?

Ask yourself...

- Is the code clean?
- Is the code tested?
- Is there a learning objective or event?
- Is there a plan for payback?
- Is the business truly informed?

TECHNICAL DEBT?

If you say no to even one...

- Is the code clean?
- ~~Is the code tested?~~
- Is there a learning objective or event?
- Is there a plan for payback?
- ~~Is the business truly informed?~~

... then you don't have Technical Debt

MESS (NOUN)

- Disorderly accumulation, heap, or jumble
- A state of embarrassing confusion
- An unpleasant or difficult situation

CRUFT (NOUN)

- An unpleasant substance
- The result of shoddy construction
- Redundant, old or improperly written code

BUT, IT'S JUST SEMANTICS

IT'S NOT JUST SEMANTICS

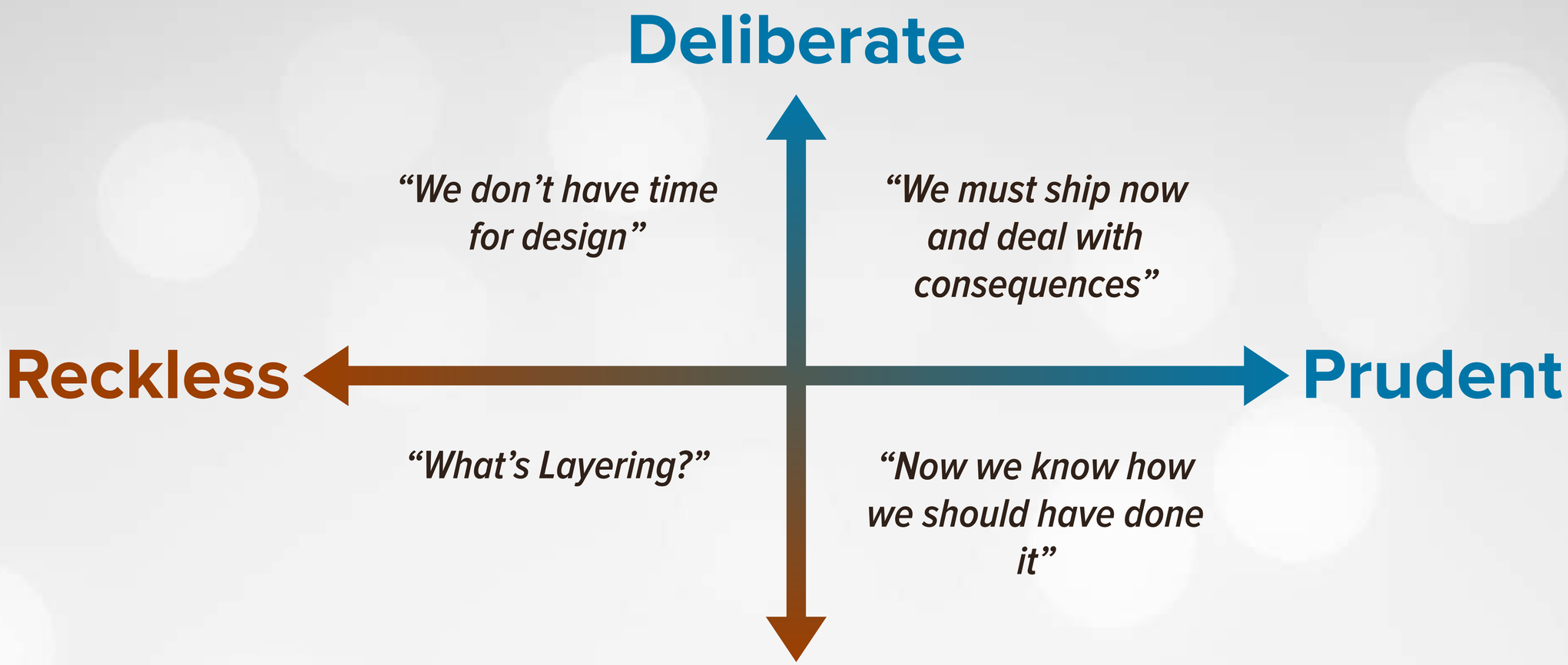
Technical Debt is Good

IT'S NOT JUST SEMANTICS

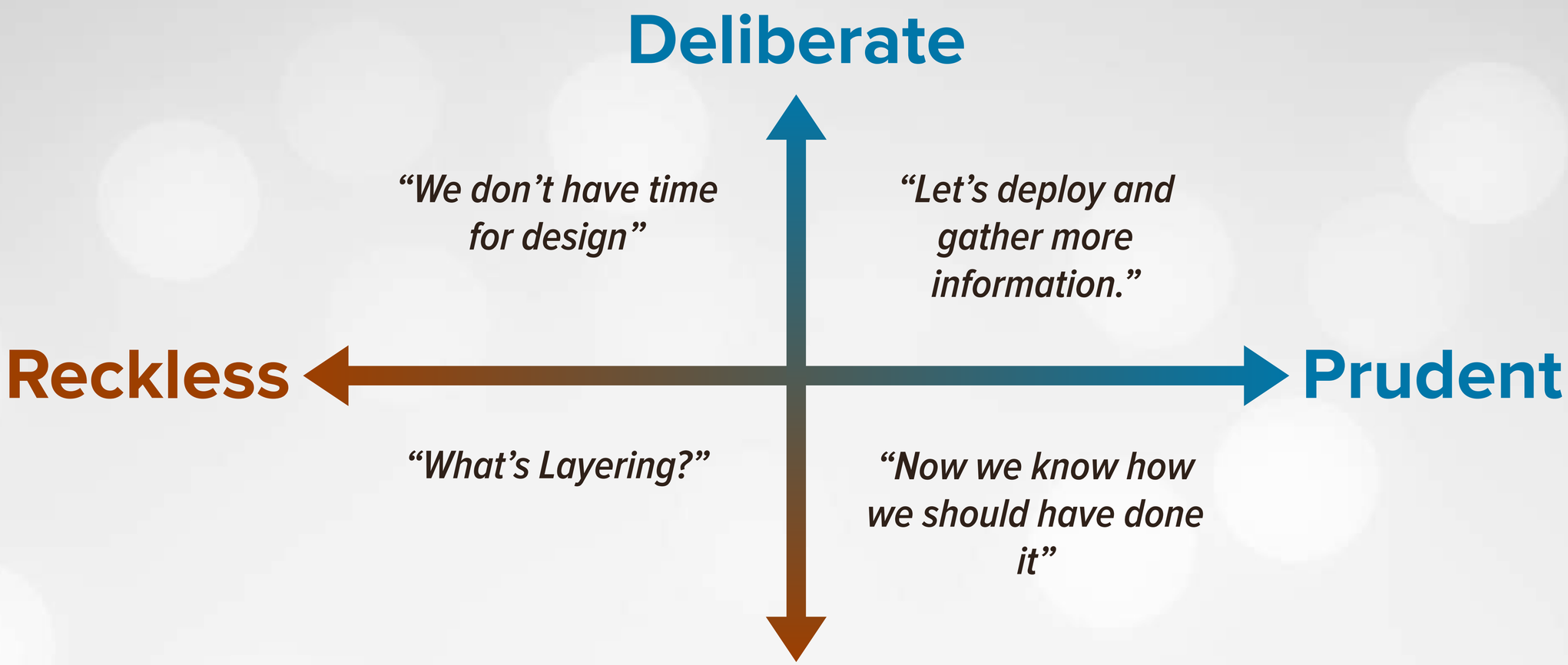
Quick and Dirty is Technical Debt is Good

IT'S NOT JUST SEMANTICS

~~Quick and Dirty is Good~~



TECHNICAL DEBT QUADRANT



TECHNICAL DEBT QUADRANT

"TECHNICAL DEBT"

in other fields

“TECHNICAL DEBT”

in other fields

“TECHNICAL DEBT”

in other fields

"TECHNICAL DEBT"

in other fields

CTO2
TALENT + TECHNOLOGY

Deliberate

RECKLESS AND DELIBERATE

"We don't have time for design"

"Let's deploy and gather more information."

Reckless

Prudent

RECKLESS AND INADVERTENT

"What's Layering?"

"Now we know how we should have done it"

Inadvertent

TECHNICAL DEBT QUADRANT

CTO2 TALENT + TECHNOLOGY

Deliberate

"We don't have time for design"

"Let's deploy and gather more information."

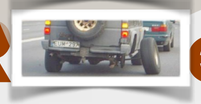
RECKLESS

PRUDENT

"What's layering?"

"Now we know how we should have done it"

Inadvertent



IRRESPONSIBLE

INCOMPETENT

TECHNICAL DEBT

TECHNICAL DEBT QUADRANT

CRUFT OR DEBT?

```
DataSet aDs, qDs;
aDs = _dbConnector.UpdateAgentList();
qDs = _dbConnector.GetQueueList();
foreach (DataTable aTable in aDs.Tables) {
  foreach (DataRow aRow in aTable.Rows) {
    foreach (DataC
      DataSet as
      foreach (D
        foreach
          foreach
            foreach (DataTable qTable in qDs.Tables) {
              foreach (DataRow qRow in qTable.Rows) {
                foreach (DataColumn qColumn in qTable.Columns) {
                  DataSet sqDs = _dbConnector.GetSkillsForQueue(qRow[qColumn].ToString());
                  foreach (DataTable sqTable in sqDs.Tables) {
                    foreach (DataRow sqRow in sqTable.Rows) {
                      foreach (DataColumn sqColumn in sqTable.Columns) {
                        foreach (string skill in sqRow[sqColumn].ToString().Split(paramDelimStr)) {
                          if (skill == asRow[asColumn].ToString()) {
                            try {
                              _dbConnector.SetAgentQueueSkill(aRow[aColumn].ToString(),
                                qRow[qColumn].ToString(),
                                skill);
                            }
                            catch { continue; }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

CRUFT OR DEBT?

```
DataSet aDs, qDs;
aDs = _dbConnector.UpdateAgentList();
qDs = _dbConnector.GetQueueList();
foreach (DataTable aTable in aDs.Tables) {
  foreach (DataRow aRow in aTable.Rows) {
    foreach (DataColumn aColumn in aTable.Columns) {
      DataSet asDs = _dbConnector.GetAgentSkills(aRow[aColumn].ToString());//AgentId
      foreach (DataTable asTable in asDs.Tables) {
        foreach (DataRow asRow in asTable.Rows) {
          foreach (DataColumn asColumn in asTable.Columns) {
            foreach (DataTable qTable in qDs.Tables) {
              foreach (DataRow qRow in qTable.Rows) {
                foreach (DataColumn qColumn in qTable.Columns) {
                  DataSet sqDs = _dbConnector.GetSkillsForQueue(qRow[qColumn].ToString());
                  foreach (DataTable sqTable in sqDs.Tables) {
                    foreach (DataRow sqRow in sqTable.Rows) {
                      foreach (DataColumn sqColumn in sqTable.Columns) {
                        foreach (string skill in sqRow[sqColumn].ToString().Split(paramDelimStr)) {
                          if (skill == asRow[asColumn].ToString()) {
                            try {
                              _dbConnector.SetAgentQueueSkill(aRow[aColumn].ToString(),
                                qRow[qColumn].ToString(),
                                skill);
                            }
                            catch { continue; }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

```
DataSet aDs, qDs;
aDs = _dbConnector.UpdateAgentList();
qDs = _dbConnector.GetQueueList();
foreach (DataTable aTable in aDs.Tables) {
  foreach (DataRow aRow in aTable.Rows) {
    foreach (DataColumn aColumn in aTable.Columns) {
      DataSet asDs = _dbConnector.GetAgentSkills(aRow[aColumn].ToString());//AgentId
      foreach (DataTable asTable in asDs.Tables) {
        foreach (DataRow asRow in asTable.Rows) {
          foreach (DataColumn asColumn in asTable.Columns) {
            foreach (DataTable qTable in qDs.Tables) {
              foreach (DataRow qRow in qTable.Rows) {
                foreach (DataColumn qColumn in qTable.Columns) {
                  DataSet sqDs = _dbConnector.GetSkillsForQueue(qRow[qColumn].ToString());
                  foreach (DataTable sqTable in sqDs.Tables) {
                    foreach (DataRow sqRow in sqTable.Rows) {
                      foreach (DataColumn sqColumn in sqTable.Columns) {
                        foreach (string skill in sqRow[sqColumn].ToString().Split(paramDelimStr)) {
                          if (skill == asRow[asColumn].ToString()) {
                            try {
                              _dbConnector.SetAgentQueueSkill(aRow[aColumn].ToString(),
                                qRow[qColumn].ToString(),
                                skill);
                            }
                            catch { continue; }
                          }
} } } } } } } } } } } } }
```

```
DataSet aDs, qDs, asDs, sqDs;
aDs = _dbConnector.UpdateAgentList();
qDs = _dbConnector.GetQueueList();

foreach (DataRow aRow in aDS.Tables[0].Rows) {
  String agentID = aRow[“AgentId”].ToString();
  asDs = _dbConnector.GetAgentSkills(agentID);
  foreach (DataRow asRow in asDs.Tables[0].Rows) {
    String agentSkill = asRow[“Skill”].ToString();
    foreach (DataRow qRow in qDs.Tables[0].Rows) {
      queueName = qRow[“QueueName”].ToString();
      sqDs = _dbConnector.GetSkillsForQueue(queueName);
      foreach (DataRow sqRow in sqDs.Tables[0].Rows) {
        foreach (string skill in sqRow[“Skills”].ToString().Split(paramDelimStr)) {
          if (skill == agentSkill) {
            try { _dbConnector.SetAgentQueueSkill(agentID, queueName, skill); }
            catch { continue; }
} } } } } }
```

```
AgentList agents = new AgentList(_dbConnector.UpdateAgentList());
QueueList queues = new QueueList(_dbConnector.GetQueueList());

foreach (Agent agent in agents) {
  foreach (Skill agentSkill in agent.skills) {
    foreach (Queue queue in queues) {
      foreach (Skill queueSkill in queue.skills.Where(x => x == agentSkill)) {
        try {_dbConnector.SetAgentQueueSkill(agent.agentID, queue.name, agentSkill); }
        catch { continue; }
      }
    }
  }
}
```

CRUFT OR DEBT?

```
if ((customer.state == “AL” && customer.type ==
      CustomerType.GENERAL_AGENT && customer.revenue > 100000)
      || (customer.type == CustomerType.RETRO_AGENT &&
         (customer.state == “WI” || customer.state == “IL”))
      || (customer.type == CustomerType.FED_MANAGEMENT &&
         customer.revenue > 150000)) { ... }
```

```
// If customer is Federally Regulated
if ((customer.state == “AL” && customer.type ==
     CustomerType.GENERAL_AGENT && customer.revenue > 100000)
     || (customer.type == CustomerType.RETRO_AGENT &&
        (customer.state == “WI” || customer.state == “IL”))
     || (customer.type == CustomerType.FED_MANAGEMENT &&
        customer.revenue > 150000)) { ... }
```

```
if (customer.isFederallyRegulated()) { ... }
```

CRUFT OR DEBT?

```
double getSpeed() {
  switch (_type) {
    case EUROPEAN:
      return getBaseSpeed();
    case AFRICAN:
      return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
    case NORWEGIAN_BLUE:
      return (_isNailed) ? 0 : getBaseSpeed(_voltage);
  }
  throw new RuntimeException ("Should be unreachable");
}
```

```
class Swallow ...
  double getSpeed() { return getBaseSpeed(); }
end class

class EuropeanSwallow ...
end class

class AfricanSwallow ...
  double getSpeed() { return super.getSpeed - coconutLoad(); }
  double coconutLoad() { return getLoadFactor() * _numberOfCoconuts; }
end class

class NorwegianSwallow ...
  double getSpeed() { return (_isNailed) ? 0 : getBaseSpeed(_voltage); }
end class
```

CRUFT IS A BAD DECISION

- You are a professional developer
- You’re going to create unintentional cruft
- You have to clean up the existing cruft

THE TRAP

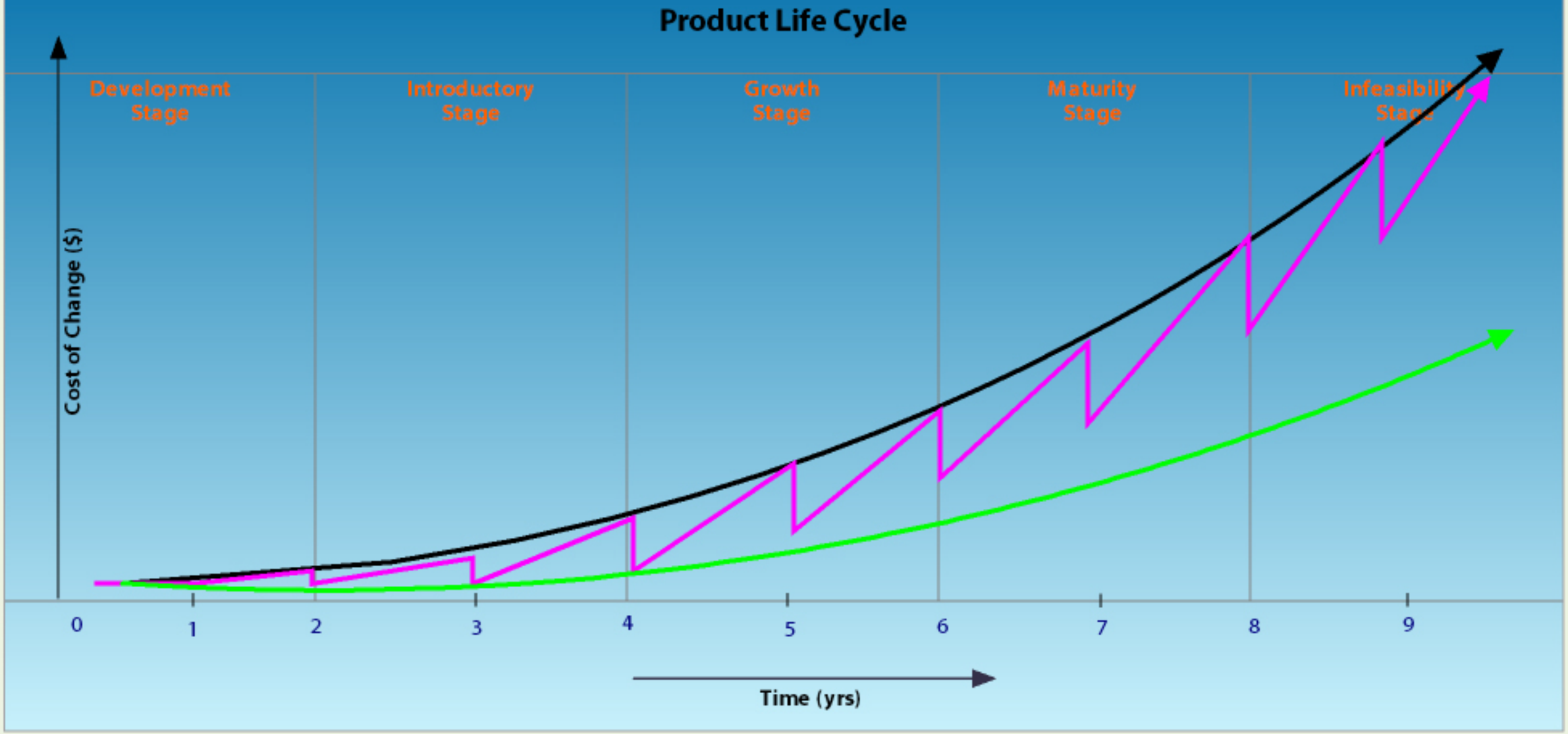
- Precedent for speed over quality
- Expectation of increased velocity
- Cruft slows you down
- Must write more cruft to keep up
- Ask permission to do your job correctly

MANAGING CRUFT

FAILING STRATEGY

Cleaning Sprint

- Schedule Iterations for cleaning code
- Defer quality to cleaning sprint
- Focus on speed/velocity at all other times



Legend:

- Traditional Application - No Refactoring (High Cost of Change)
- Once-a-year Technical Debt Refactoring (Lower Cost of Change)
- Continuous Refactoring (Lowest Total Cost of Change)

WINNING STRATEGY

Clean Constantly

- Never make an intentional mess
- Monitor your “Technical Debt”
- Follow the Boy Scout Rule
- Remember quality is your responsibility
- NEVER ask permission to do your job correctly

MONITOR YOUR CRUFT

- Code Coverage
- Code Complexity
- Coupling
- Maintainability
- Monitor Trends, Not Points

REVIEW

Technical Debt

- A strategic design decision
- Requires business to be informed
- Includes a pay-back plan

Cruft

- Happens
- Needs to be monitored and cleaned
- Is NOT Technical Debt

REVIEW

Technical D

- A strategic design de
- Requires busines nformed
- Includes a p plan

cruft

- Ha
- to be monitored and cleaned
- NOT Technical Debt

NEVER ASK PERMISSION TO DO YOUR JOB CORRECTLY

THANK YOU!

#etka17 / #TechnicalDebt / @DocOnDev / @WeAreCTO2

DOC NORTON
CO-FOUNDER + CEO
DOC@WEARECTO2.COM

FOR MORE

Send a blank email to
cto2@SendYourSlides.com
with the subject line:
TechnicalDebt

DOC NORTON
CO-FOUNDER + CEO
DOC@WEARECTO2.COM

TECHNICAL DEBT TRAP