# Why is Software Engineering so difficult?

James Miller

# Scale and Complexity

- Real Software Engineering problems are:
  - Huge …. Vast volumes of code, documentation, ….
  - Complex …. Vast volumes of people, many different objectives,…
- University Software Engineering problems are:
  - Small and straight-forward
- This clearly causes a problem in appreciating
  - What is really going on; and
  - What the problems really are

# Scale and Complexity

Windows 7 > 50 million LOC

***Expect a staggering number of bugs.***

# Bugs?

- Well-written C and C++ code contains some 5 to 10 errors per 100 LOC after a clean compile, but before inspection and testing.

- At a 5% rate any 50 MLOC program will start off with some 2.5 million bugs.

# Bug removal

- Testing typically exercises only half the code.
- It’s hard to devise tests that check rarely-invoked exception handlers, deeply nested IFs and nested loops.
- So the 50% test coverage number suggests that Windows 7 shipped with 1.25 million bugs.

# Better bug removal?

- *"There are better ways to do testing that do produce fantastic programs."*

- Are we sure about this fact?
  - No, its only an opinion!
  - In general Software Engineering has ….

***NO FACTS!***

- Code coverage, for instance, can insure every branch and conditional has been taken. It's required by the FAA's DO-178B level A standard for safety-critical avionics.

# So why not do this?

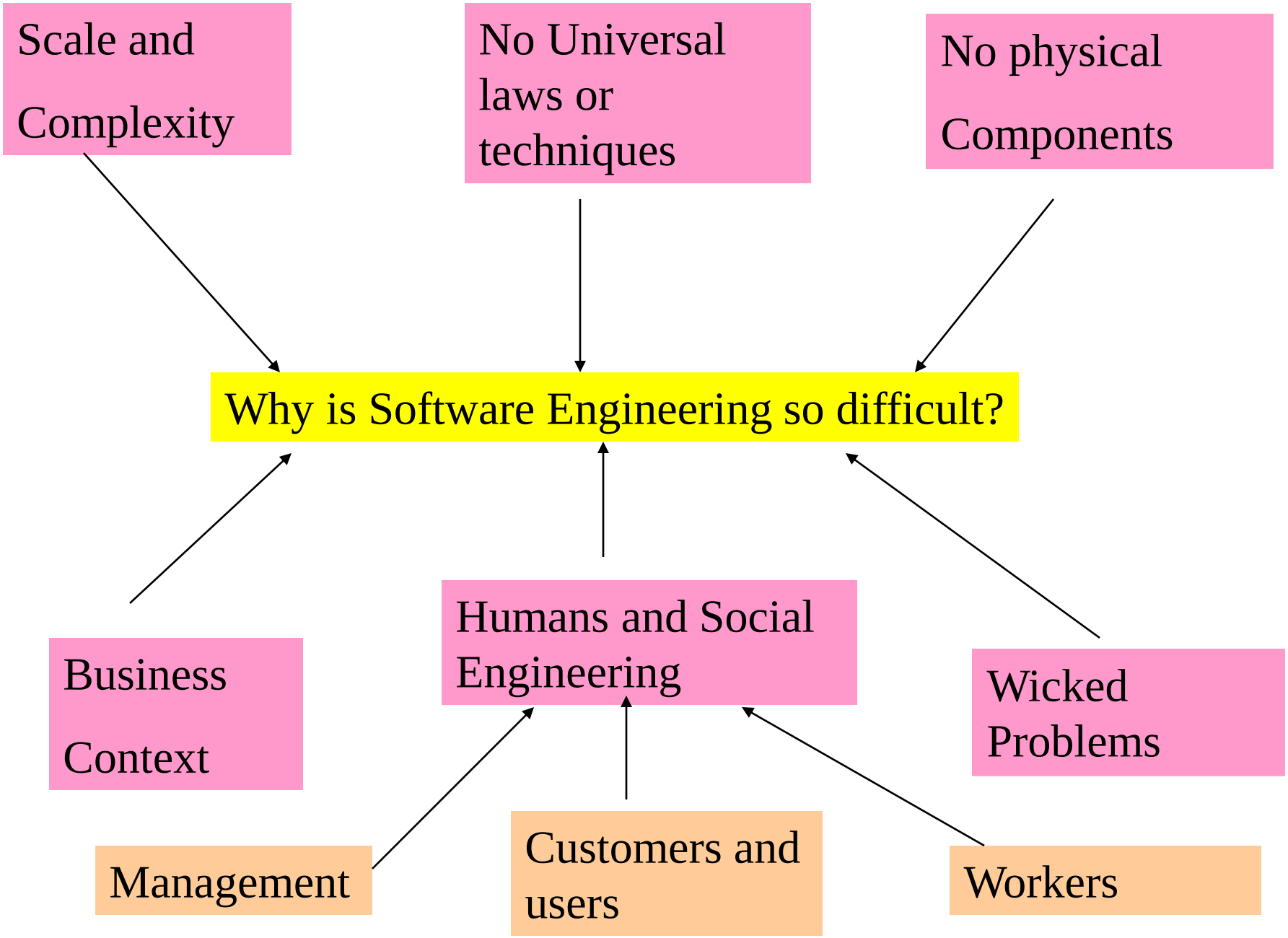
- The costs are unbelievable.
- It's not unusual for the qualification process to produce a half page of documentation for each line of code.
- A 50 MLOC program's doc might be:
  - 25 million pages long,
  - consuming 50,000 reams of paper
  - a stack 2 miles high.
- Will Windows 8 undergo this evaluation? No.

# Are embedded systems better?


***Programs are gigantic today, and will be simply unbelievable tomorrow.***

Is that the entire story?

No!


Scale and Complexity
No Universal laws or techniques
No physical Components
Why is Software Engineering so difficult?
Business Context
Humans and Social Engineering
Wicked Problems
Management
Customers and users
Workers

# Wicked Problems

Why mathematics can’t solve everything; and why simple procedural techniques don’t work!

# Problems….

- … can be split into two types
- Tame problems – typically the sort of thing dealt with in classes.
- Wicked problems – commonly found in many real-world settings especially within the IT industry.
- Wicked problems have 6 characteristics.

# *1. You don't understand the problem until you have developed a solution.*

Every solution that is offered exposes new aspects of the problem, requiring further adjustments of the potential solutions. Indeed, there is no definitive statement of "The Problem."

***The problem is ill structured, an evolving set of interlocking issues and constraints.***

- One cannot understand the problem without knowing about its ***context***;
- one cannot meaningfully search for information without ideas about the solution.
- ***one cannot first understand, then solve.***
- Moreover, what “the Problem” is depends on who you ask – *different stakeholders* have *different views* about what the problem is and what constitutes an acceptable solution.

# *2. Wicked problems have no stopping rule.*

- Since there is no definitive “The Problem”, there is also no definitive “The Solution.”
- The problem solving process ends when you run out of resources, such as time, money, or energy, not when some optimal or “final and correct” solution emerges.
- Herb Simon, Nobel laureate in economics, called this “satisficing” -- stopping when you have a solution that is “good enough”

# 3. *Solutions to wicked problems are not right or wrong.*

- They are simply “better,” “worse,” “good enough,” or “not good enough.”
- With wicked problems, the determination of solution quality is not objective and cannot be derived from following a formula.
- Solutions are assessed in a social context in which:
  - “many parties are equally equipped, interested, and/or entitled to judge [them],”
  - and these judgments are likely to vary widely and depend on the stakeholder’s independent values and goals.

# 4. *Every wicked problem is essentially unique and novel.*

- There are so many factors and conditions, all embedded in a dynamic social context, that make no two wicked problems alike….
- ***solutions to them will always be custom designed and fitted.***
- "The condition in a city constructing a subway may look similar to the conditions in Edmonton, … but differences in commuter habits or residential patterns may far outweigh similarities in subway layout, downtown layout, etc"
- ***Over time one acquires wisdom and experience about the approach to wicked problems, but one is always a beginner in the specifics of a new wicked problem.***

# 5. *Every solution to a wicked problem is a "one-shot operation."*

- Every attempt has consequences.
- "One cannot build a freeway to see how it works."
- This is the "Catch 22" about wicked problems: you can't learn about the problem without trying solutions, but every solution you try is expensive and has lasting unintended consequences which are likely to spawn new wicked problems.

# 6. *Wicked problems have no given alternative solutions.*

- There may be no solutions, or there may be a host of potential solutions that are devised, and another host that are never even thought of.
- Thus, it is a matter of *creativity* to devise potential solutions,
- and a matter of *judgment* to determine which are valid, which should be pursued and implemented.
- These criteria are more descriptive than definitional.

- Here are a few examples of wicked problems:

  - Whether to route the highway through our city or around it?

  - How to deal with crime and violence in our schools?

  - What to do when oil resources run out?

  - What should our mission statement be?

  - ***What features should be in our new product?***

# Wicked Problem Example: A New Car Design

- Let's consider a potentially wicked problem in the design of a new car.
- Let’s imagine a project team that has formed around a new assignment:

  *the Marketing department is asking for a design that emphasizes side-impact safety – they want to promote a new “safe car” to compete with Volvo.*

- The problem to be solved is the work of the project.
- There is a deadline, a budget and a senior executive that the project reports to….

# *1. You don't understand the problem until you have developed a solution.*

- One approach to making a safer car would be to add structural supports in the doors.
- It turns out that the additional door structure:
  - doubles the cost of the door,
  - makes the doors heavier and harder to open and close,
  - changes the fuel mileage and ride,
  - and requires an adjustment to the suspension and braking systems.
- ***Making the doors stronger leads into other design problems.***

- It also bounces back into marketing problems such as:

  - "What should the price be?",

  - "How much do people really care about side impact survivability?",

  - "What do customers really want in a car?"
- ***All of these problems interact with each other.***
- And at the senior executive level, the real question is:

  *"Should we continue to produce this new car?"*

# *2. Wicked problems have no stopping rule.*

- When does the car become “safe”?
- There is no natural stopping point in working out the tradeoffs among safety, performance, appearance, and cost.
- At some point, the design team will be forced to make a decision.
- If it were not for project deadlines, the team would swirl indefinitely in “analysis paralysis”.

# *3. Solutions to wicked problems are not right or wrong.*

- No amount of study, laboratory experiments, or market surveys will establish that that project team's solution is "correct."
- Ironically, when the car gets produced:
  - there will be reviews pointing out that the doors are heavy and difficult to open when parking on a hill,
  - mixed with law suits from people who were injured in side-impact accidents despite the stronger doors.

# *4. Every wicked problem is essentially unique and novel.*

- Even if the project team has several successful car designs under its belt,
- the “safe door” problem is essentially unique and novel, because of the configuration of issues and stakeholders.

- Consider the following external actions which have just happened….

1. A recent study by a consumer safety organization suggests that side impact injuries would be reduced by side air bags, which are not a part of the design.
2. A side-impact injury lawsuit has been filed against the company

   – if the new design is announced now, it may look like an acknowledgement of prior unsafe designs.
3. Moreover, federal legislation is emerging that may put legal constraints on the strength of the doors.

***The design of safer doors is not merely a technical problem: it is a political and PR problem as well.***

# *5. Every solution to a wicked problem is a "one-shot operation".*

- When the new safer car finally reaches the market, it may be a flop, or it may change the safety standards for the whole industry.
- The design team can build prototypes of the car and test them, but there is no way to anticipate the unintended consequences of producing and selling the new vehicle.

# *6. Wicked problems have no given alternative solutions.*

- The safe door problem does not have a few discrete possible solutions from which to choose.
- There is an immense space of options in terms of structural reinforcement, materials, cushioning, window design, hinge placement, and how the door latches and opens.
- The design team cannot select from a few options – it must collectively exercise creativity and judgment about an elegant resolution of all of the design priorities.

# The design of a new “safe car” is an example of a wicked problem.

- It cannot be solved by engineers alone,
- nor is there any way of determining that any given solution is “correct” or even optimal.
- And the perceived quality of the solution varies from stakeholder to stakeholder.

# Why is Software Engineering so difficult?

## Implications

# No silver bullet!

- No software engineering tool, technique or approach can universally solve the entire problem or even a sub-problem.
- Different contexts demand different approaches.
- Hence, we end up learning lots of different tools, techniques and approaches which seem to solve the same problem.

# Which is best?

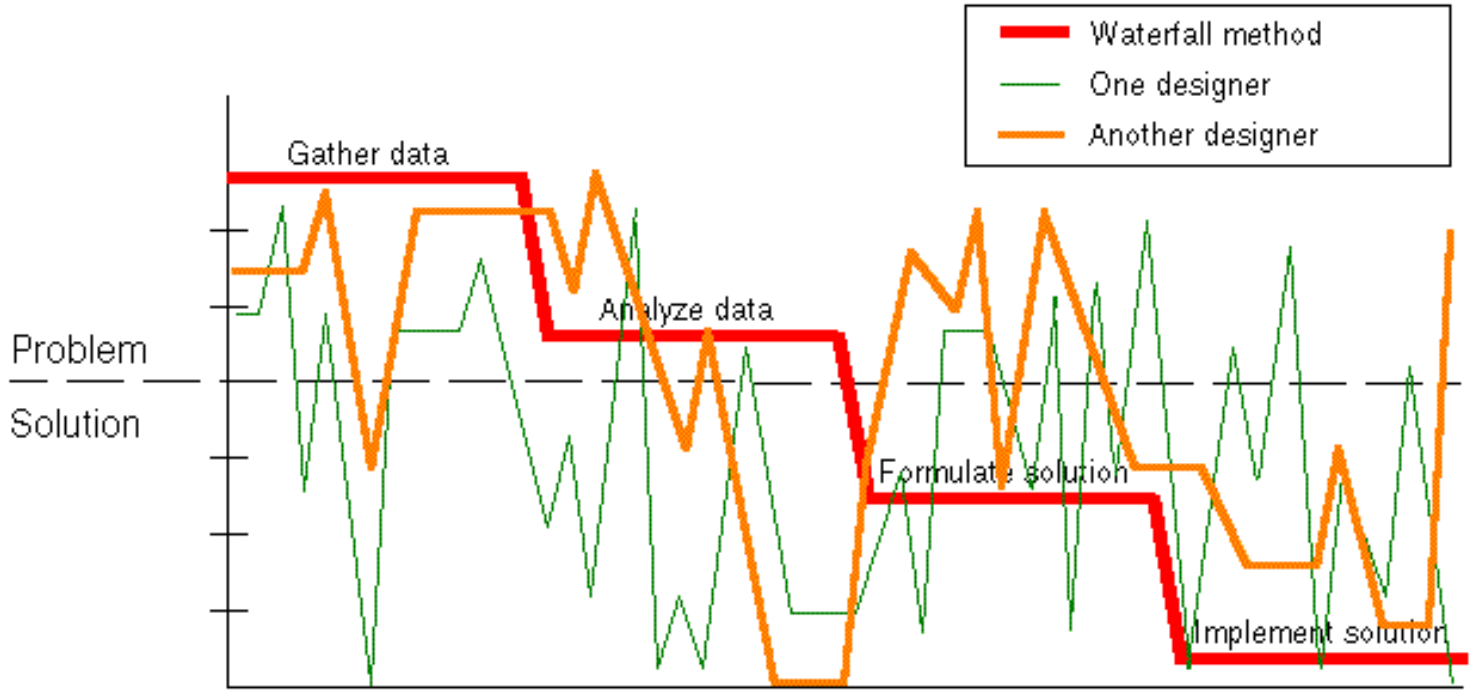
- Is technique A better than technique B?

- A really difficult question to answer.
- Normal answers are
  - it depends…. ***On the context!***
  - Or “who knows”

# “The devil is in the details….”

- Software Engineering approaches are normally described at a high level.
- Why: to allow a single description.
- A technique is actually a family of techniques
  - the details of the technique normally change with the context.
  - Commonly, the Software Engineer has to create the details of the technique during its usage.

# Linear approaches don't work



# Sequencing is often fatal!

- In class we learned apply techniques:

  A -> B -> C -> D

- So when I use A, this means I always do

  B -> C -> D

- Wrong!
  - If use are using A, its because you think the context demands it.
  - The context says nothing about B,C, and D

# The Software Engineer learns

- A big bag of tricks or a quiver full of arrows.
- The context, or problem, is the target.

- Given a specific target, the software engineer tries to select their best arrows from their quiver to maximize their score at shooting at this target.

- Unfortunately, remember, nobody is really sure about how to calculate scores in this game!