Lean Software Development: Introduction. Week 1

Giancarlo Succi, Artem Kruglov

Innopolis University

g.succi@innopolis.ru
a.kruglov@innopolis.ru

January 12, 2023

Engineering the production of software systems

We often make analogies between producing code and cooking...

Poll!

What is the purpose of software engineering?

Ideas

My Idea of the Software Engineer

Background on Lean Management

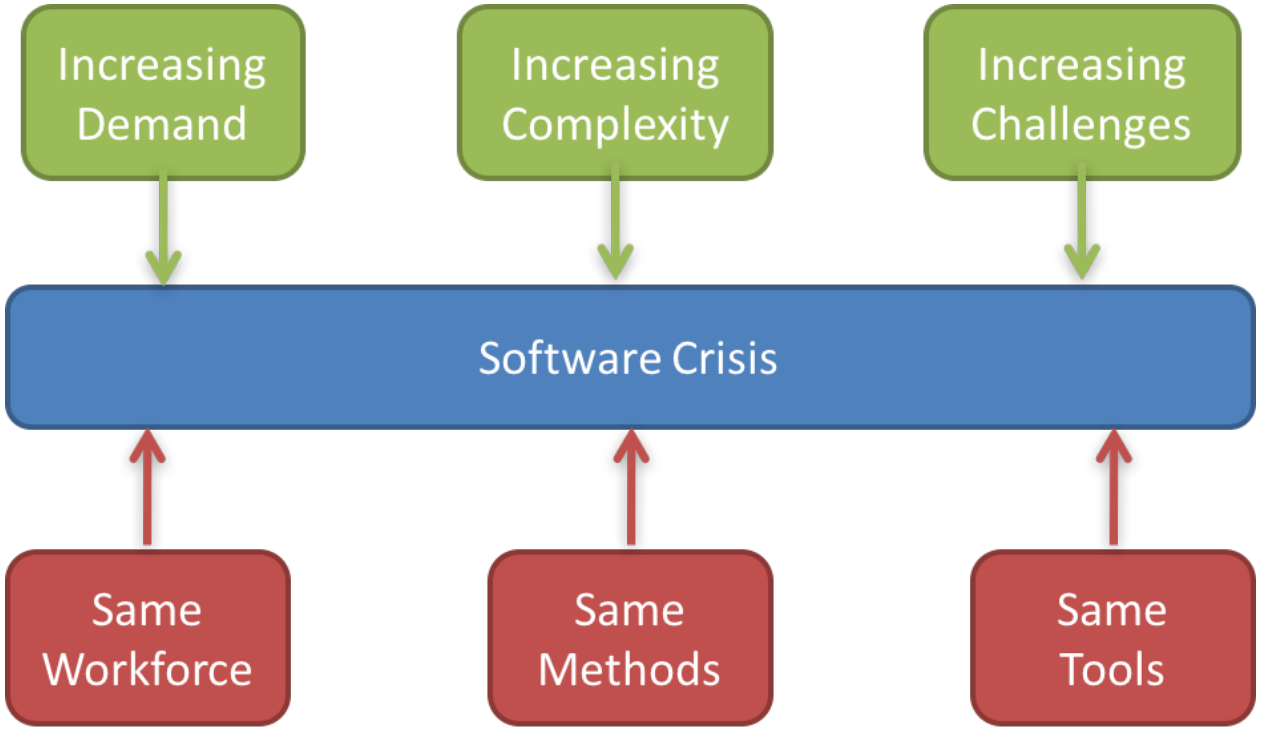
The “Software Crisis”

The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem. Edsger Dijkstra, 1972

http://qualityandprogramming.blogspot.ru/2012/03/crisis-del-software.html

The "Software Crisis"

Consecutive revolutions are being achieved in many domains thanks to computer software



However, software engineering is not seeing the same evolution rate as in the domains it addresses.

http://www.slideshare.net/rui_curado/abse-and-atomweaver-a-quantum-leap-

The “Software Crisis”

The causes of the software crisis were linked to the overall complexity of hardware and the software development process. The crisis manifested itself in several ways:

- Projects running over-budget
- Projects running over-time
- Software was very inefficient
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code difficult to maintain
- Software was never delivered

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

Software errors in history - 1

NASA's Mars Climate Orbiter – On its mission to Mars in 1998 the Climate Orbiter spacecraft was ultimately lost in space. Although the failure bemused engineers for some time, it was revealed that a subcontractor on the engineering team failed to make a simple conversion from English units to metric.
An embarrassing lapse that sent the **$125 million** craft fatally close to Mars' surface after attempting to stabilize its orbit too low. Flight controllers believe the spacecraft ploughed into Mars' atmosphere where the associated stresses crippled its communications, leaving it hurtling on through space in an orbit around the sun.

Take with modifications from:
https://raygun.com/blog/2014/05/10-costly-software-errors-history/

Software errors in history - 2

Ariane 5 Flight 501 – Europe's newest unmanned satellite-launching rocket reused working software from its predecessor, the Ariane 4. Unfortunately, the Ariane 5's faster engines exploited a bug that was not found in previous models. Thirty-six seconds into its maiden launch the rocket's engineers hit the self-destruct button following multiple computer failures. In essence, the software had tried to cram a 64-bit number into a 16-bit space. The resulting overflow conditions crashed both the primary and backup computers (which were both running the exact same software).
The Ariane 5 had cost nearly **$8 billion** to develop, and was carrying a **$500 million** satellite payload when it exploded.

Take with modifications from:
https://raygun.com/blog/2014/05/10-costly-software-errors-history/
Video: https://youtu.be/qnHn8W1Em6E

The "Software Crisis"

The challenges of building software stem from our intrinsic lack of understanding it and the essential difficulties of building software can be divided into the following four categories:

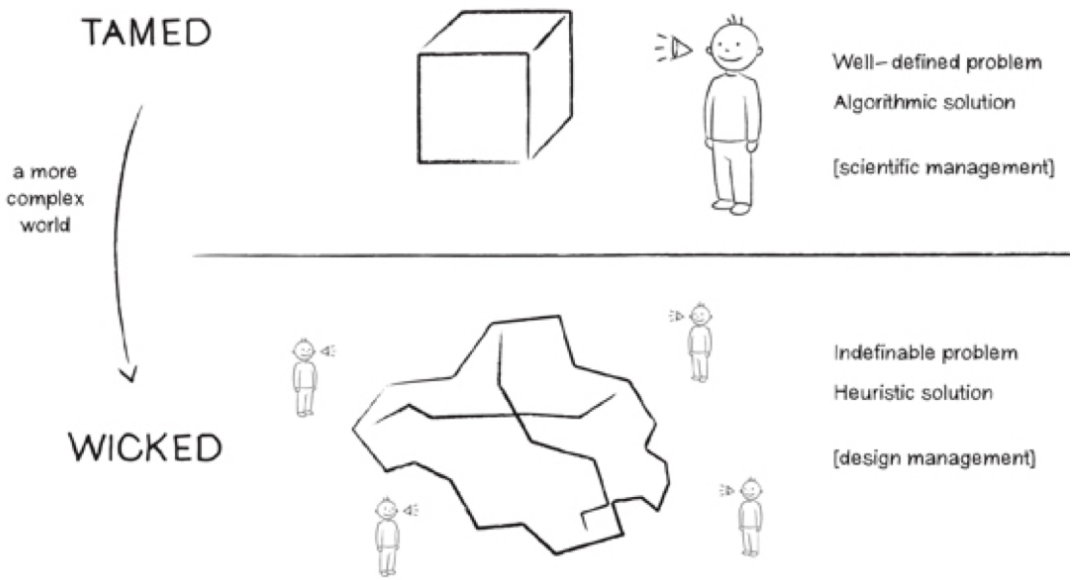
- **Complexity:** software systems consist if a multiple of unlike parts that can assume a large number of different states. This makes it difficult to conceive, describe, and test software;
- **Conformity:** software has to be integrated with previously developed software systems and laws and conform to the way people are used to work. These dependencies increase the likelihood that the software has to be changed in the future;

The “Software Crisis”

- **Changeability:** all successful software gets changed - as new uses of a software product are found, stakeholders push for modifications. Moreover, successful software survives the life cycle of the environment (e.g., operating system, hardware, etc.) it was written for. This means software has to be updated to the new needs or environments it has to run it;
- **Invisibility:** the invisibility of software and the difficulties in visualizing it make it difficult to reason and to communicate about it.

Tame and Wicked Problems

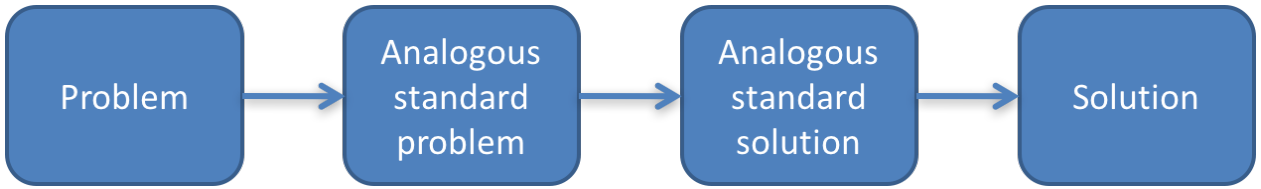
Tame problems are those problems that can be easy engineered; they can be formulated exhaustively and stated containing all the information needed for understanding and solving the problem.
Not all problems are tame. There are also **wicked problems**. The information needed to describe the wicked problem depends on the idea to solve it.



https://vivifychangecatalyst.wordpress.com/2015/06/28/wicked-problems-innovation-and-project-masiluleke/

Tame Problems

- Has a well-defined and stable problem statement.
- Has a definite stopping point, i.e., when the solution is reached.
- Has a solution which can be objectively evaluated as right or wrong.
- Belongs to a class of similar problems which are all solved in the same similar way.
- Has solutions which can be easily tried and abandoned.
- Comes with a limited set of alternative solutions



http://www.slideshare.net/curtistim/wicked-issues-taming-problems-and-systems

Wicked Problems

- Wicked projects cannot provide a definitive, analytical formulation of the problem they target. Formulating the project and the solution is essentially the same task. Each time you attempt to create a solution, you get a new, hopefully better, understanding of the project.
- Wicked projects have no a stopping rule telling when the problem they target has been solved. The problem-solving process proceeds iteratively and ends when resources are depleted and/or stakeholders lose interest in a further refinement of the currently proposed solution.
- Solutions to problems in wicked projects are not true or false, but good or bad. Since there are no unambiguous criteria for deciding if the project is resolved, getting all stakeholders to agree that a resolution is "good enough" can be a challenge.

Wicked Problems

- There is no immediate or ultimate test of a solution to the targeted problem in a wicked project. Solutions to such projects generate waves of consequences, and it is impossible to know how these waves will eventually play out.
- Each solution to the problem targeted by a wicked project has irreversible consequences. Care must be placed in managing assumed solutions. Once the website is published or the new customer service package goes live, you cannot take back what was online or revert to the former customer database.
- Wicked projects do not have a well-described, widely accepted set of potential solutions. The various stakeholders may have differing views of what are acceptable solutions. It is a matter of judgment as to when enough potential solutions have emerged and which should be pursued.

Wicked Problems

- Each wicked project is essentially unique. There are no well-defined "classes" of solutions that can be applied to a specific case. It is not easy to find analogous projects, previously solved and well documented, so that their solution could be duplicated.
- The problem targeted by a wicked project can be considered a symptom of another problem. A wicked project deals with a set of interlocking issues and constraints that change over time, embedded in a dynamic and evolving context.
- The causes of a problem targeted by a wicked project can be explained in several ways. There are several stakeholders who have various and changing ideas about what is the project, its nature, its causes, and the associated solution.
- The project must not go wrong. Mistake is not an option here. Despite the inability to express the project solution analytically, it is not allowed to fail the project.

Software Development Is a Wicked Problem

- It is very hard to plan a software development project upfront considering all eventualities.
- A software product is hardly perfect or finished; as soon as users use it, new requirements will arise.
- There does not exist the one single solution to a software engineering problem.
- We cannot determine how well an implementation solves the requirements until we implement it.
- Choices are sometimes very costly to reverse. The last option is to throw away the existing product and start from scratch.

Software Development Is a Wicked Problem

- There are infinite ways to solve a software development problem.
- Every software development project is essentially unique.
- Software engineering is a wicked problem on multiple levels: choosing the "best" database system, user interface, operating system, hardware, etc. is also a wicked problem.
- The problem a software system aims to solve is seen differently by the stakeholders of the system: users, developers, maintainers, database operators, etc.
- Software development (i.e., to try to solve the problem) means to spend resources. To be wrong, i.e., to deliver a solution that the users retain not useful, is not considered an option.

How a Lean approach can address the wickedness of software

- The iterative, customer-oriented approach reduces the need of detailed planning, which reduces complexity. Moreover, it is also harder to fail the project since the client, the one that defines what "failure" means, keeps the control over the output of the project during the development.
- An approach based on measurement and collaboration helps to understand if the team is going towards the right direction and to balance the needs of different stakeholders. A flexible, Agile approach helps to keep the costs of change low which lowers the need to find the "best" solution immediately and allows to find a solution that fits the specific context.
- Just-in-time production maximizes the chances to know how well an implementation solves the requirements.
- An approach that systematically collects experience and reuses it in future projects helps to collect "best practices" for a specific context.