From Users to Behaviour Designers: Could AI agents change skills and professional roles?

From Programmed Machines to AI Agents:
blurring the boundary between
software development and software use?

Luca Mari
Università Cattolica, Milano, Italy
luca.mari@unicatt.it
https://lmari.github.io

Abstract

Software development and software use have long been distinct activities, requiring different skills and typically performed by different people. With the recent emergence of so-called AI agents, this distinction is beginning to blur: part of the work of design, automation, and adaptation is shifting from developers to users. The enabler of this transformation is technical: the availability of well-trained AI systems capable of interpreting instructions, generating artefacts, and coordinating actions. Yet its possible implications are not merely technical, as it could reshape competences, professional identities, organisational structures, and the very meaning of using digital systems. The talk will provide background, discuss firsthand examples, and develop some hypotheses and reflections on the organisational and cultural transformation we may be entering.

Luca Mari is Full Professor of measurement science in the Department of Philosophy of Università Cattolica, Milano, Italy. He teaches courses on measurement science and statistical data analysis, systems theory, and digital thinking. His research and outreach activities concern the broad context of information science and technology, ranging from fundamental topics of measurement science to dynamical systems modeling and artificial intelligence, including its human-centred and societal implications.

Two strategies for artificial behaviour

The contrast helps explain the novelty of machine learning

The “Ada strategy”

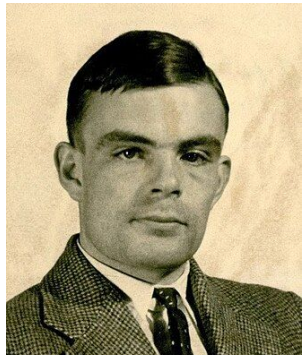
“It can do whatever we know
how to order it to perform.”

A. Ada Lovelace,
Notes on Babbage’s Analytical Engine,
1842

- Decide the wanted behaviour
- Design the procedure that realizes the behaviour
- Implement the procedure in a system
  that can execute it

Software (and any other technological artefact)
as a **programmed machine**

The “Alan strategy”



“Instead of trying to produce a programme
to simulate the adult mind,
why not rather try to produce one
which simulates the child’s?
If this were then subjected
to an appropriate course of education
one would obtain the adult brain.”

Alan Turing, Computing Machinery and Intelligence, 1950

- Decide the wanted behaviour
- Design the learning setting that can induce the behaviour
- Train the system on suitable data and constraints
- Validate the learned behaviour in relevant conditions

Software as a **learning machine**

Programmed machines execute procedures (designed by humans)
Learning machines acquire behavioural dispositions (shaped by humans)

The context

Specializing the generic and unclear concept of artificial intelligence

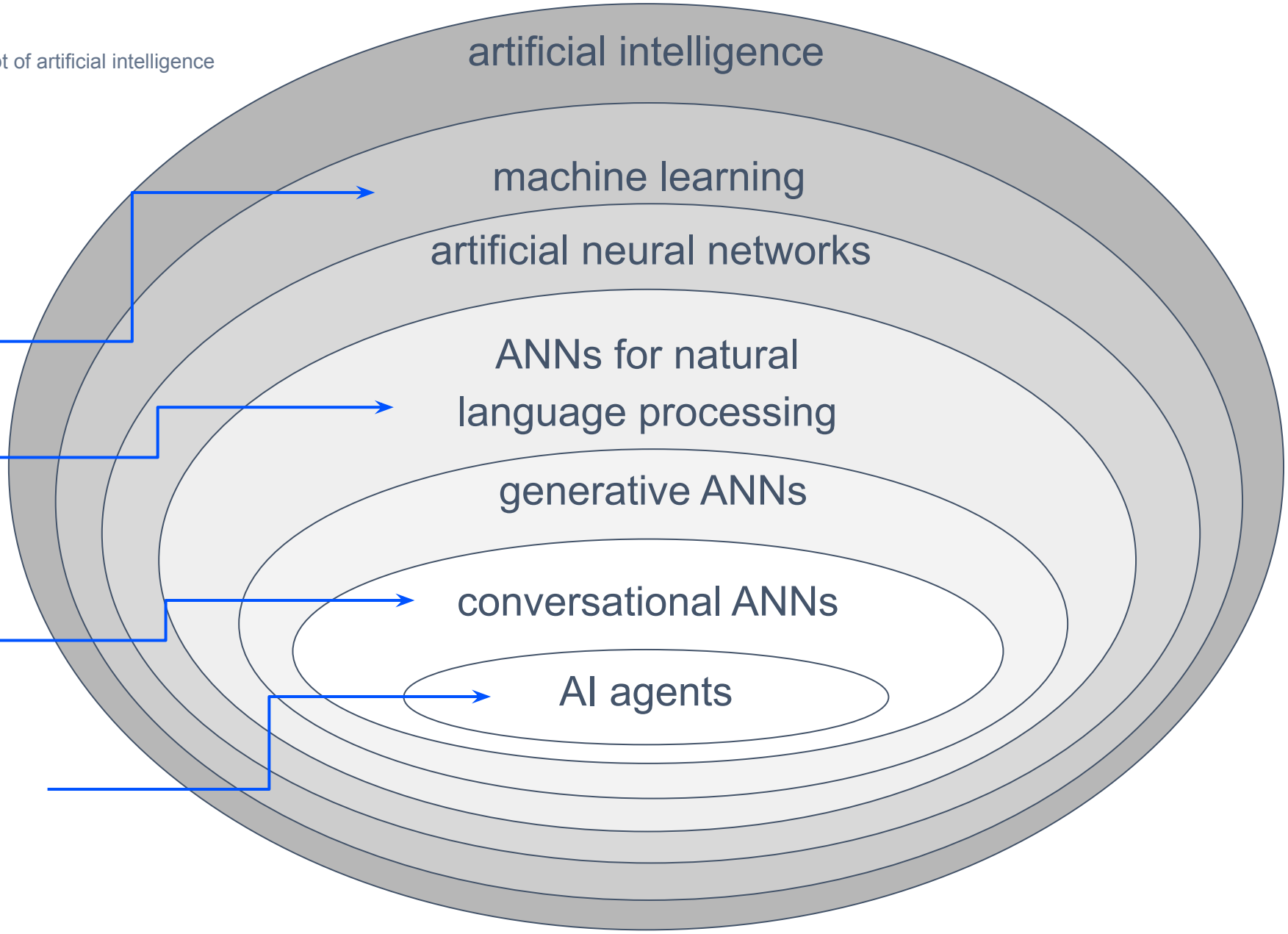
Machines that

– learn

– comunicate in our own languages

– maintain the context of the communication

– not only “speak” but also “do”



The claim

A technical evolution with organisational and cultural consequences

Software development and software use have long been distinct activities, made by individuals with different competences and purposes: AI agents are blurring the boundary

Part of the work of design, automation, and adaptation is shifting from developers to (power) users

Technical enabler

AI agents can interpret instructions, generate artefacts, coordinate actions, and use tools

Human consequence

Competences, professional identities, and organisational structures may be reshaped

Cultural question

Is the meaning of “using a digital system” changing?

Software is the currently prominent case of a more general pattern

The transition

From programmed machines to learning machines

Programmed machines

Behaviour is specified primarily
by explicit instructions:
we tell the machine what to do
and how to do

Development is
a **procedure-oriented,**
instruction-driven process

Validation typically asks:

- Is the procedure correct?
- Is it implemented correctly?
- Are edge cases covered?

Machines are in principle **white boxes**:
trust is anchored in
specification and execution

The novelty is that
the behaviour of machines
is produced through training

Learning machines

Behaviour is learned,
and not described by the programmer
as a procedure

Development is
a **task-oriented,**
output-driven process

Validation typically asks:

- What behaviour emerges after training?
- How robust is it?
- Under which conditions does it fail?

Machines are in practice **grey boxes**:
trust is anchored in
observed performance

Is it now a well-known story?

Learning machines, and chatbots in particular, are widespread

From chatbots to AI agents: more than just hype?

A key technical enabler for the change: function calling

Before (a “pure” chatbot)

User: What time is it in New York?

Chatbot: I can only provide and process information that was part of my training: I don't know, I’m sorry.

After (an “instrumented” chatbot)

System prompt: Whenever appropriate, call the function get_time() that returns…

User: What time is it in New York?

Chatbot (hidden “reasoning”): I am not able to satisfy user’s request, but I can delegate the task to an available tool.

Chatbot (hidden function call to a Python interpreter): exec get_time(location=”New York”)

Python interpreter (hidden execution output): 2026-5-17 17:53:16

Chatbot (hidden "reasoning”): The user wants to know the current time in New York, which is 2026-5-17 17:53:16: let's prepare a nicely formatted answer.

Chatbot: In New York it is 5:53 pm.

Agentic behaviour is a matter of behavioural delegation:
is it triggering a *second transition*?

A second (very recent, possible) transition

From chatbots to AI agents

Chatbots (as learning machines)

Behaviour is learned,
and not described by the
programmer
as a procedure

Development is
a **task-oriented,**
output-driven process

Validation typically asks:

- What behaviour emerges after training?
- How robust is it?
- Under which conditions does it fail?

Machines are in principle **grey boxes**:
trust is anchored in
observed performance

The novelty is that
the behaviour of machines
is contextual and operational

Agentic machines

Behaviour is learned,
and may include delegation
to both rule-based and other AI components

Development is
a **goal-oriented,**
purpose-driven process

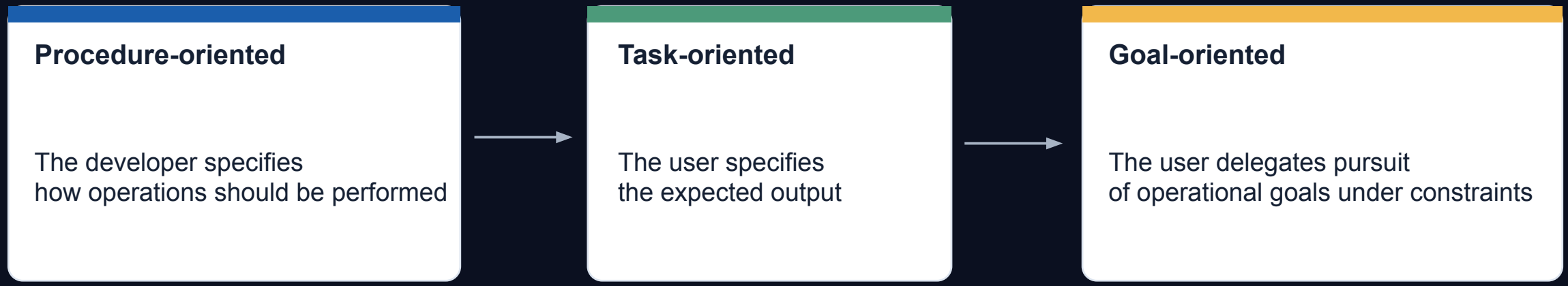
Validation typically asks:

- How does it interpret the goal?
- What plan does it choose?
- Which actions does it perform?

Machines are in practice **black boxes**:
trust is anchored in
controlled delegation

A model of two transitions

Things are changing so rapidly…



The earlier layers do not disappear: they become embedded in higher levels of delegation

In task-oriented systems, the user delegates producing an output
In goal-oriented systems, the user delegates a trajectory of behaviour

This is where using a system begins to resemble configuring a (more or less bounded) assistant

Do we have time for discussing an example? (some lessons learned)

Let's make it concrete

Some features of the activity performed by the AI agent (GPT Codex 5.x within VS Code):

- ~30k lines of code entirely generated from natural language prompts
- full life cycle of development (coding, testing, debugging, deployment, maintenance) handled in automatic way
- multi-language user interface and project documentation generated and updated in automatic way

Some features of the interaction with the AI agent:

- it understands requests remarkably well, and the quality of the generated code is usually excellent
- it autonomously tests the produced code and often identifies and fixes problems on its own
- it maintains coherence across extensions and modifications of existing components, including refactoring when needed
- it can discuss alternative solutions, suggest next development steps, and sometimes proposes useful extensions
- it handles complex requests as multi-part tasks, often decomposed into incremental development stages
- its explanations are generally clear, technically sound, and well aligned with the performed work
- it produces and continuously updates high-quality documentation from the codebase
- it interacts like an expert developer, including practical reasoning about implementation strategies and risks
- it is able to operate the software it has generated, for example by creating valid data files in the expected format

Some limitations of the example:

- single user / no team development process
- no application-specific libraries / codebase
- widely known languages (JavaScript, CSS, HTML, YAML)

Moving on...

What about some next steps?

Can we identify some possible consequences of these transitions?

Three interaction formulas

What we specify changes

Procedure-oriented

Typical formula:
“Do these steps.”

Main object of design:
• instructions
• workflows
• explicit rules

Primary success criterion:
correct execution

Task-oriented

Typical formula:
“Produce this output.”

Main object of design:
• prompts
• examples
• expected results

Primary success criterion:
adequate output

Goal-oriented

Typical formula:
“Pursue this goal.”

Main object of design:
• goals
• constraints
• priorities
• stopping conditions

Primary success criterion:
appropriate delegated behaviour

Is English (or Italian, or…) becoming the new programming language?
(“vibe coding” and well beyond…)

Validation shifts with delegation

From checking execution to auditing delegated behaviour

Validate execution

Did the system follow the specified procedure correctly?

Methods: code inspection, unit tests, formal verification, debugging

Validate output

Is the produced result adequate for the assigned task?

Methods: benchmarks, test sets, human evaluation, gold standards

Validate behaviour

Was the whole trajectory appropriate under goals, context, and constraints?

Methods: traceability, policy checks, action audits, human oversight

Procedure-oriented systems require validation of execution
Task-oriented systems require validation of outputs
Goal-oriented systems require validation of delegated behaviour

The new failure mode

Efficiently pursuing the wrong goal

The more behaviour we delegate, the more critical goal specification becomes

Procedure failure

The steps are wrong or poorly implemented

Task failure

The output is inaccurate, incomplete, or unsuitable

Goal failure

The agent interprets or pursues the goal in a way that violates context, constraints, or values

A capable agent can make a bad specification more consequential

Required user’s competences: are they changing?

User’s competence changes shape

Procedure writer

Knows how to specify procedures and data structures

Task requester

Knows how to formulate requests, assess outputs, and iterate with the system

Behaviour designer

Knows how to define goals, constraints, risks, priorities, checkpoints, and stopping conditions

The emerging skill is not “using AI instead of thinking”,
but designing the conditions under which delegated behaviour should unfold

Professional roles: a boundary is beginning to blur

Some design work migrates from development to use

Developer

Traditionally designs functions, workflows, data structures, automation logic

User

Traditionally operates functions and consumes or produces outputs

**With agentic systems,
the user may increasingly specify:**

- goals and acceptable trade-offs
- tool-use permissions and limits
- evaluation criteria
- desired behavioural style

**This is not the end of development:
it is the redistribution of design**

Organisational transformation

Agentic AI changes where responsibility and control must be placed

Workflows

From fixed workflows
to adaptive workflows
partly assembled at run time

Key question:
what should remain fixed,
and what may be delegated?

Governance

From tool adoption
to (more or less) bounded
autonomy management

Key question:
who defines permissions,
checkpoints, audit trails,
and escalation?

Professional identity

From tool users
to designers
of socio-technical behaviour

Key question:
what does expertise mean
when agents can perform
parts of expert work?

The question is not simply “Can AI agents do it?”,
but “Under what organisational conditions should they be allowed to do it?”

A working summary: procedure-oriented → task-oriented → goal-oriented systems

Two transitions toward systems
able to operationalise human purposes
into goals, plans, actions,
monitoring criteria, and adaptive behaviour

The design of controlled delegation
could become a key competence
and a critical responsibility
for users becoming developers

Designing for controlled delegation: for discussion

Some questions for an organisation adopting AI agents

Goal	What exactly is the agent allowed to pursue?
Constraints	Which legal, ethical, economic, and quality limits are non-negotiable?
Tools	Which systems may it access, read, write, or modify?
Checkpoints	When must it ask for confirmation or stop?
Traceability	Can we reconstruct what it did, why, and with what evidence?
Responsibility	Who remains accountable for delegated action?

Agentic systems require not only better prompts, but better delegation contracts

Thanks for your attention

Luca Mari
luca.mari@unicatt.it
https://lmari.github.io