Assuring the Future of Software Engineering & AI Engineering

Starting Point: The SEI's Study on Future of Software Engineering

- CMU SEI's National Agenda Study (November 2021) was intended to catalyze the software engineering community by creating a research & development vision, strategy, & roadmap to engineer the next-generation of software-reliant systems

DARPA

NSF

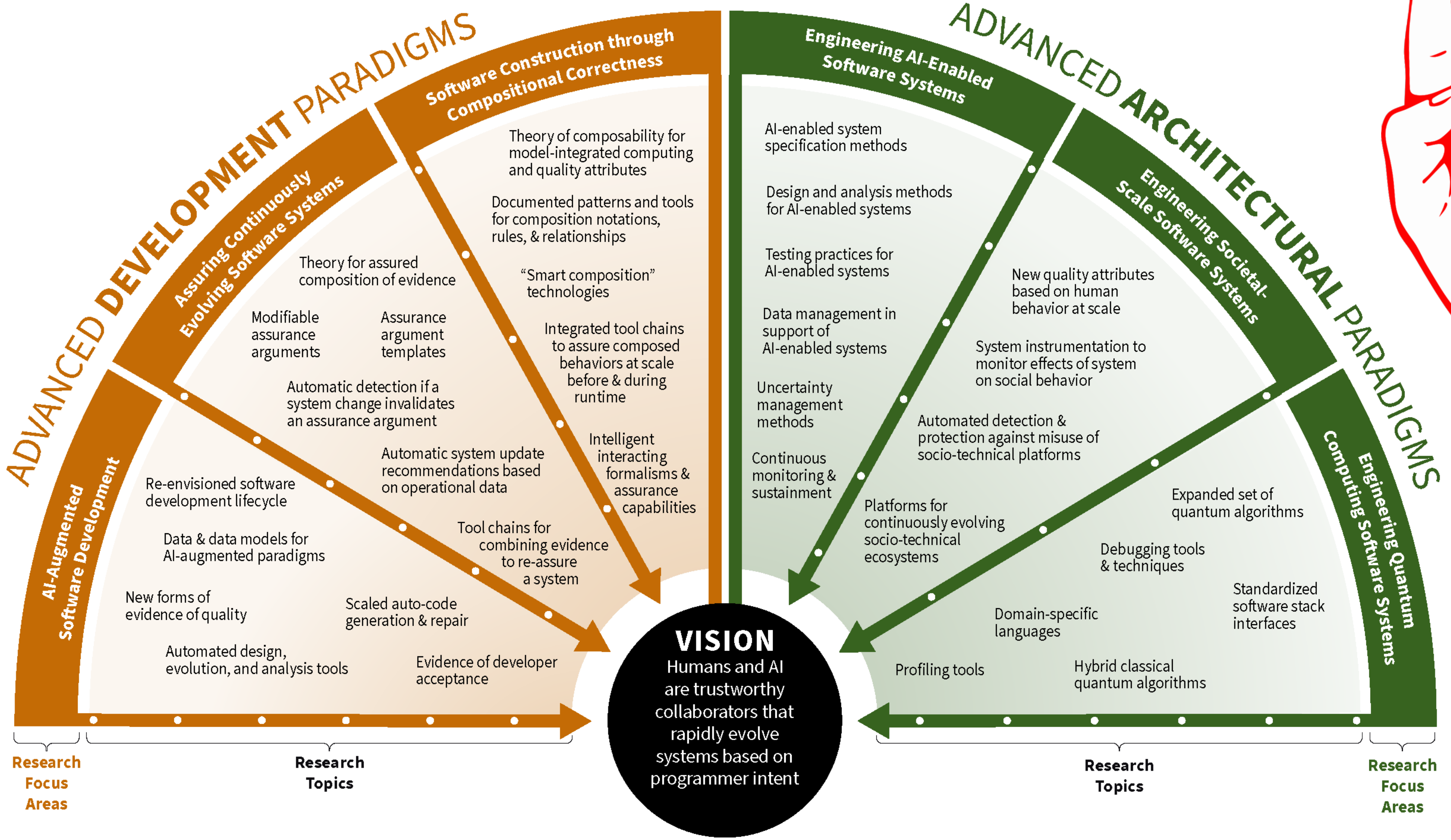
Software Engineering Institute

Carnegie Mellon

Study available at www.sei.cmu.edu/go/national-agenda

The Study Defined a Software Engineering Roadmap for 10-15 Years

- The software engineering roadmap codified research focus areas & research objectives

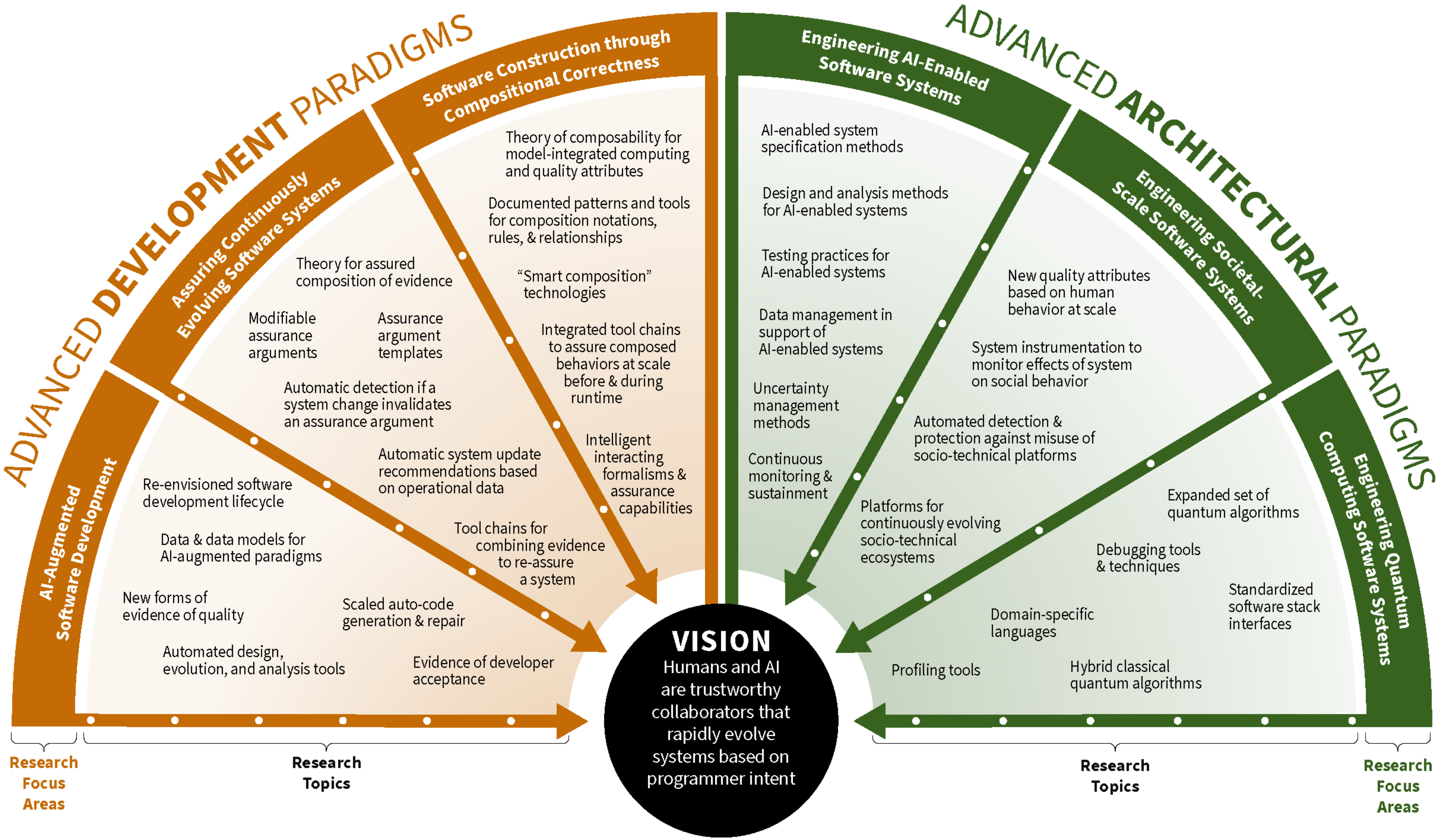


"Predictions are hard, especially about the future" – Niels Bohr & Yogi Berra

The Study's Emerging Vision of the Future of Software Engineering

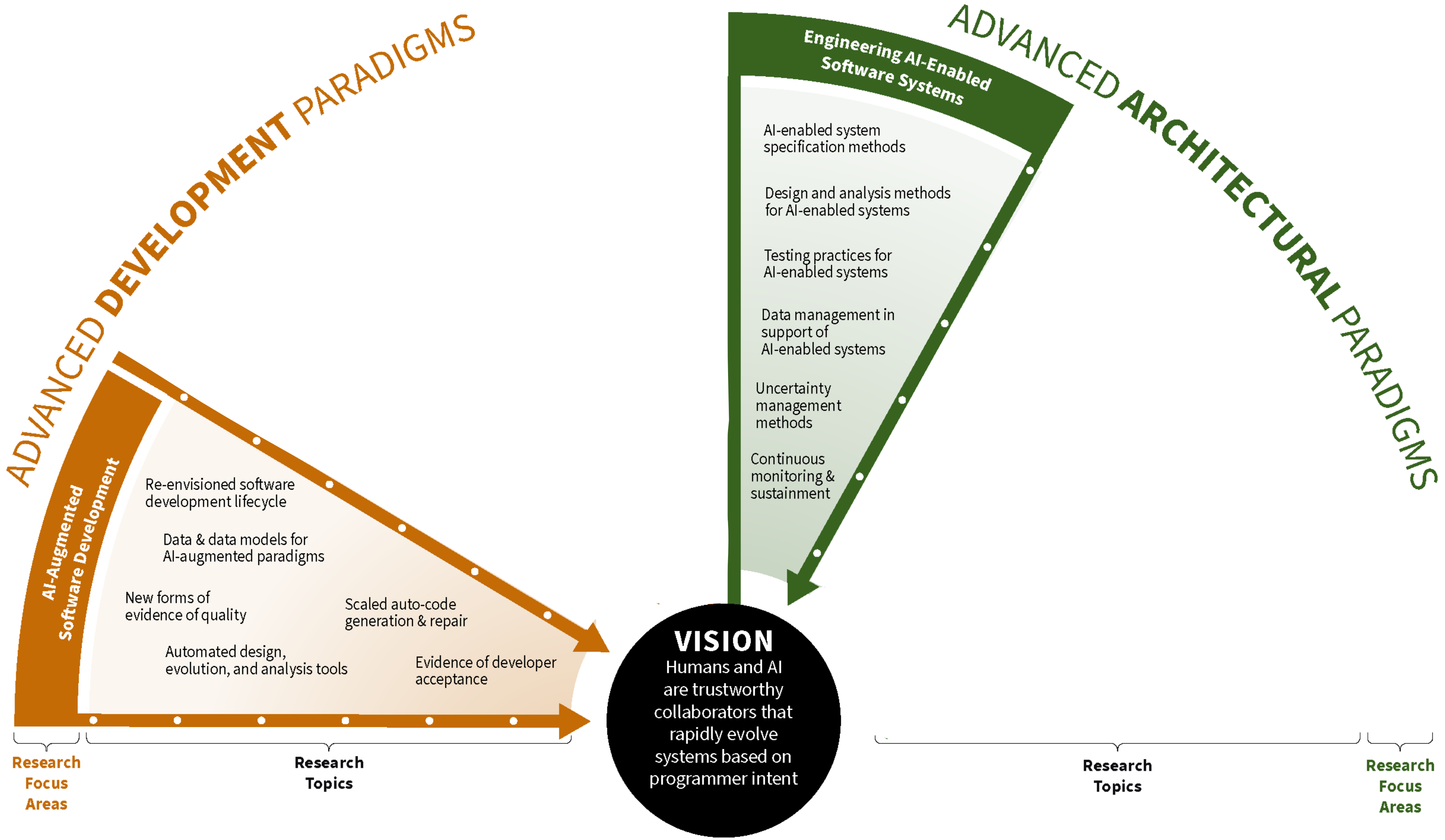
- "The current notion of software development will be replaced by one where ***the software pipeline consists of humans & AI as trustworthy collaborators that rapidly evolve systems based on programmer intent***"

How Advances in Generative AI are Affecting Our Study Findings



Our original study covered six research focus areas

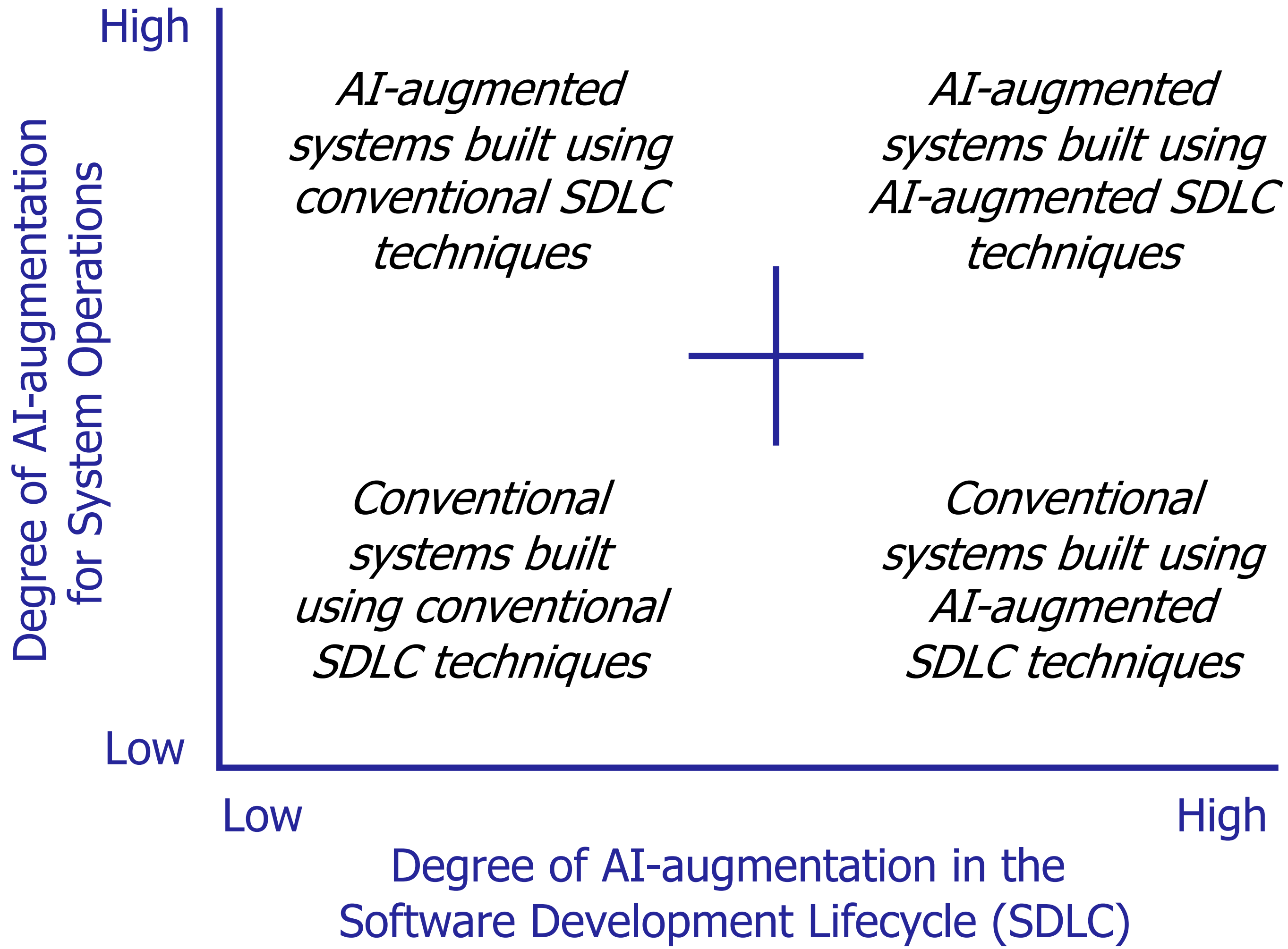
How Advances in Generative AI are Affecting Our Study Findings



Two of these six focus areas dealt with AI-augmentation for development & operations

How Advances in Generative AI are Affecting Our Study Findings

- Based on recent experience, we've created a new taxonomy of the degree of AI-augmentation for system operations & for the software development lifecycle (SDLC)



Application of Large Language Models (LLMs) in Software Engineering: Overblown Hype or Disruptive Change?

IPEK OZKAYA, ANITA CARLETON, JOHN E. ROBERT, AND DOUGLAS SCHMIDT (VANDERBILT UNIVERSITY)

OCTOBER 2, 2023

Has the day finally arrived when large language models (LLMs) turn us all into better software engineers? Or are LLMs creating more hype than functionality for software development, and, at the same time, plunging everyone into a world where it is hard to distinguish the perfectly formed, yet sometimes fake and incorrect, code generated by artificial intelligence (AI) programs from verified and well-tested systems?

LLMs and Their Potential Impact on the Future of Software Engineering

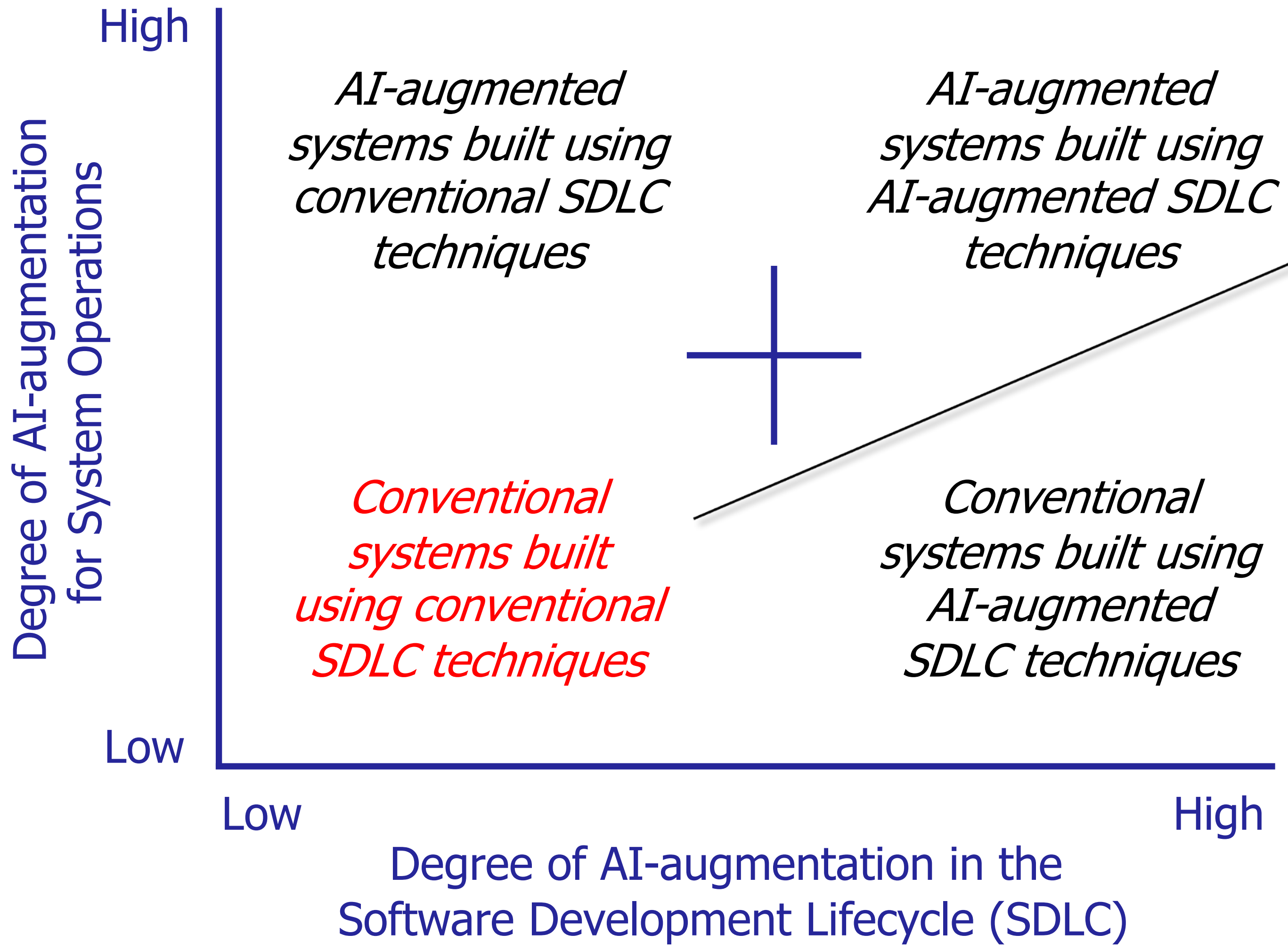
This blog post, which builds on ideas introduced in the IEEE paper *Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications* by Ipek Ozkaya, focuses on opportunities and cautions for LLMs in software development, the implications of incorporating LLMs into software-reliant systems, and the areas where more research and innovations are needed to advance their use in software engineering. The reaction of the software engineering community to the accelerated advances that LLMs have demonstrated since the final quarter of 2022 has ranged from snake oil to no help for programmers to the end of programming and computer science education as we know it to revolutionizing the software development process. As is often the case, the truth lies somewhere in the middle, including new opportunities and risks for developers using LLMs.

Research agendas have anticipated that the future of software engineering would include an AI-augmented software development lifecycle (SDLC), where both software engineers and AI-enabled tools share roles, such as copilot, student, expert, and supervisor. For example, our November 2021 book *Architecting the Future of Software Engineering: A National Agenda for Software Engineering Research and Development* describes a research path toward humans and AI-enabled tools working as trusted collaborators. However, at that time (a year before ChatGPT was released to the public), we didn't expect these opportunities for collaboration to emerge so rapidly. The figure below, therefore, expands upon the vision presented in our 2021 book to codify the degree to which AI augmentation can be applied in both system operations and the software development lifecycle (Figure 1), ranging from conventional methods to fully AI-augmented methods.

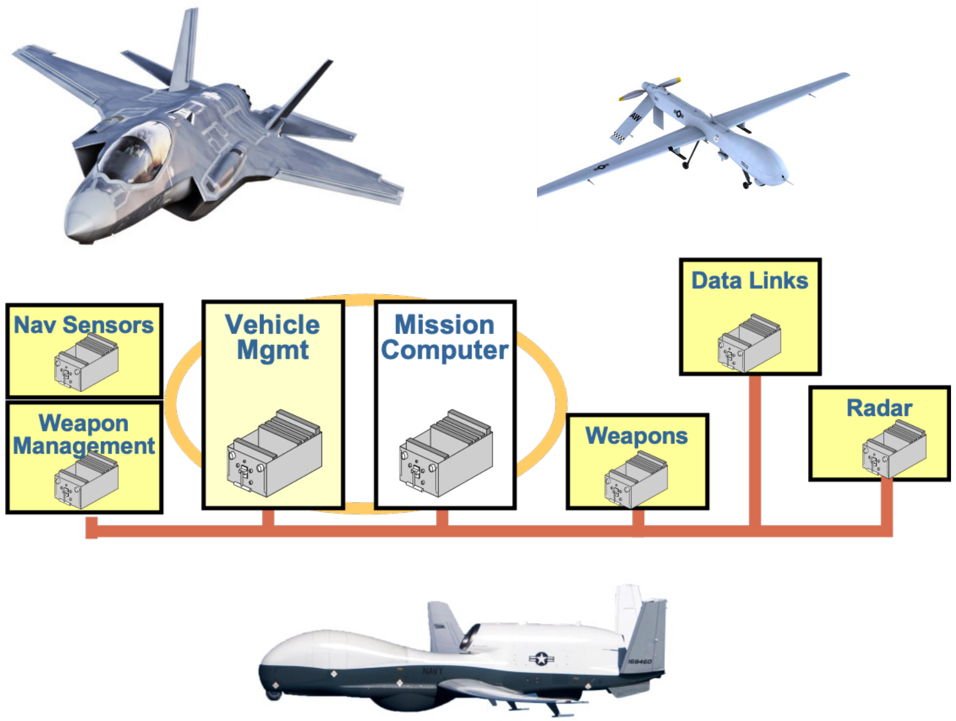
See application-of-large language-models-llms-in-software-engineering-overblown-hype-or-disruptive-change

How Advances in Generative AI are Affecting Our Study Findings

- Based on recent experience, we've created a new taxonomy of the degree of AI-augmentation for system operations & for the software development lifecycle (SDLC)



An avionics mission computing system developed w/conventional SDLC techniques with no AI-augmented tools or methods



See www.dre.vanderbilt.edu/~schmidt/corba-research-realtime.html

How Advances in Generative AI are Affecting Our Study Findings

- Based on recent experience, we've created a new taxonomy of the degree of AI-augmentation for system operations & for the software development lifecycle (SDLC)

Degree of AI-augmentation for System Operations: High / Low

AI-augmented systems built using conventional SDLC techniques

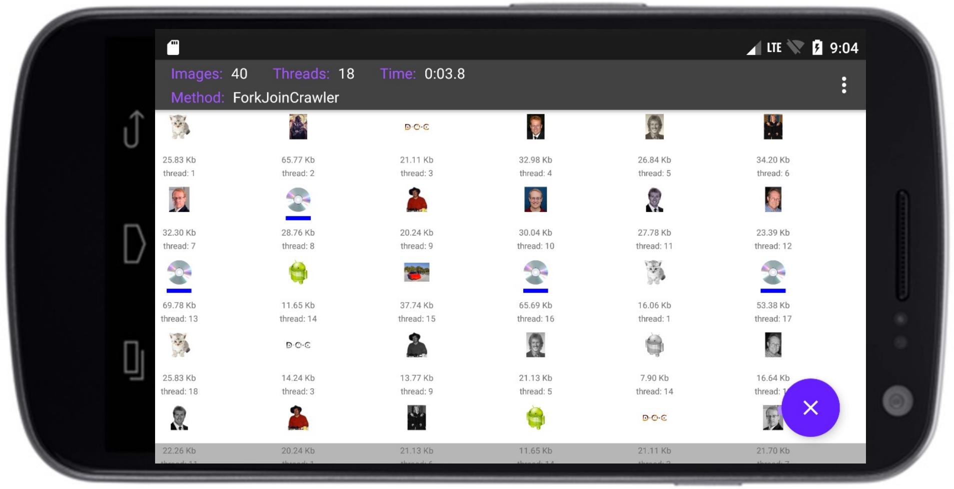
AI-augmented systems built using AI-augmented SDLC techniques

Conventional systems built using conventional SDLC techniques

Conventional systems built using AI-augmented SDLC techniques

Degree of AI-augmentation in the Software Development Lifecycle (SDLC): Low / High

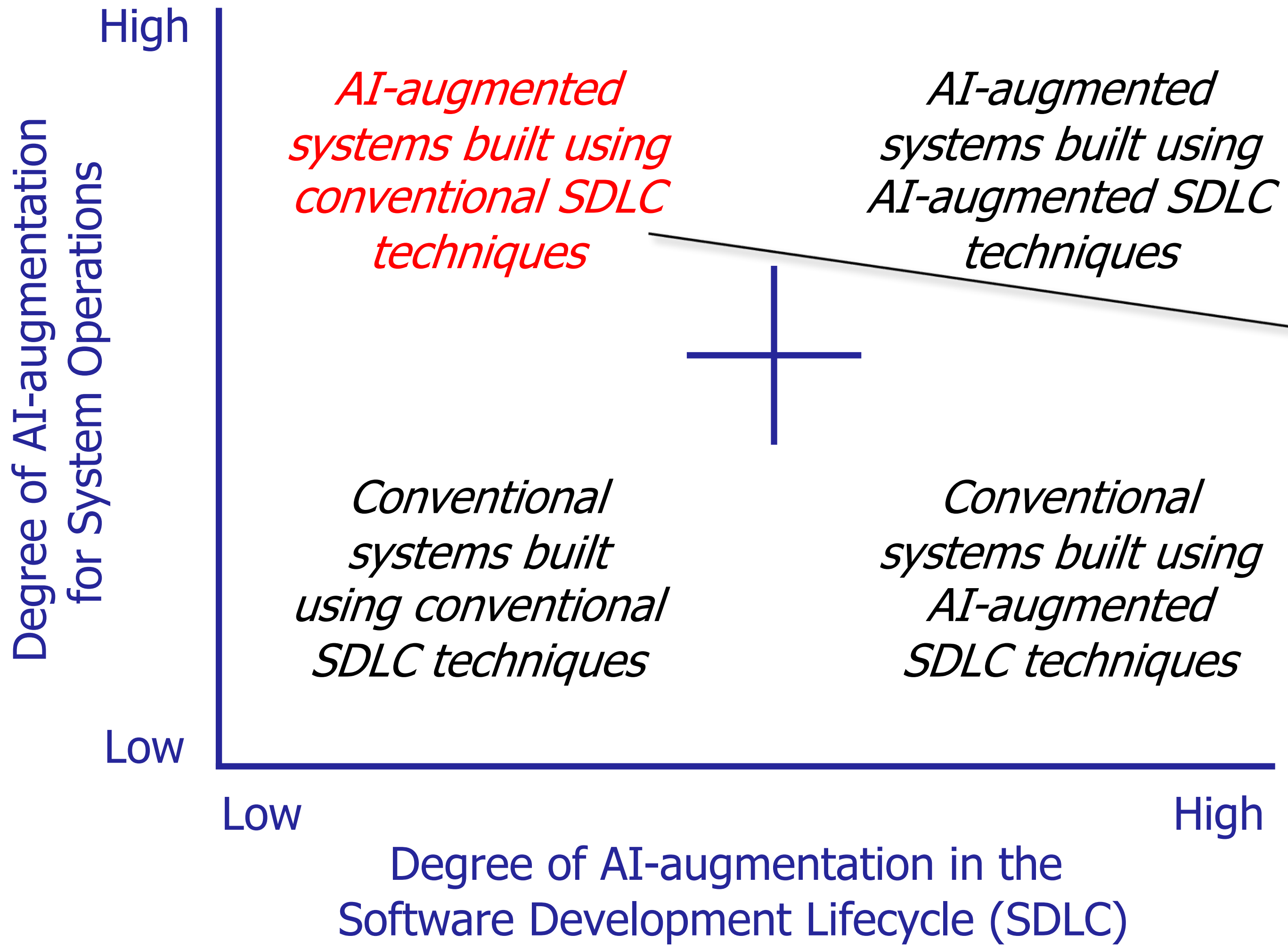
A mobile web crawler app where the logic & content is not AI-augmented, although the SDLC process applies AI-augmented code reviews, code generators, and/or testing tools



See www.youtube.com/watch?v=18TzQM6Yu9s

How Advances in Generative AI are Affecting Our Study Findings

- Based on recent experience, we've created a new taxonomy of the degree of AI-augmentation for system operations & for the software development lifecycle (SDLC)

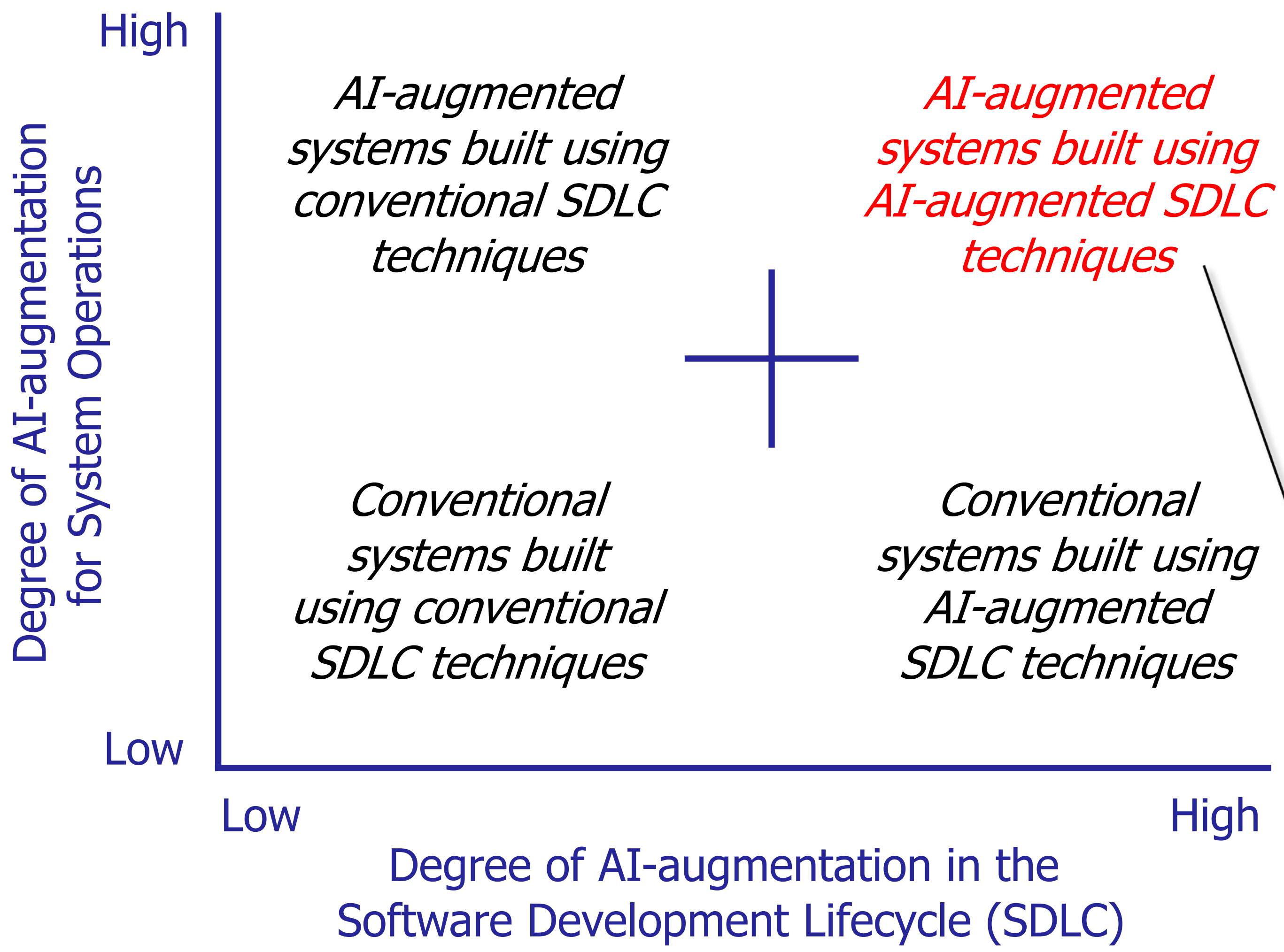


A recommendation engine in an e-commerce platform that employs machine learning for customized recommendations, however, the software itself is developed, tested, & deployed using Agile methods

See www.youtube.com/watch?v=_d_cEVpGOrg

How Advances in Generative AI are Affecting Our Study Findings

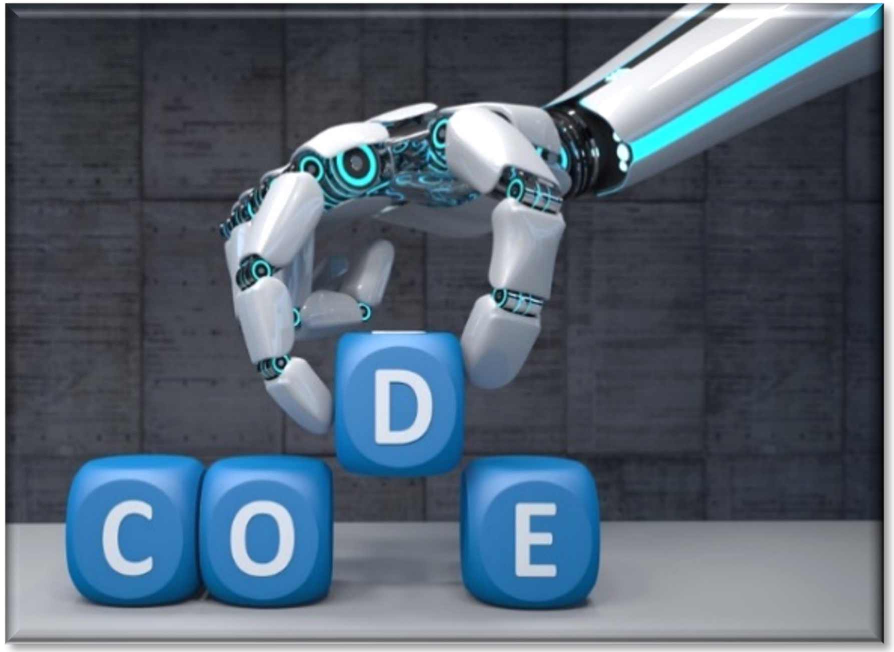
- Based on recent experience, we've created a new taxonomy of the degree of AI-augmentation for system operations & for the software development lifecycle (SDLC)



A self-driving car system that uses machine learning algorithms for navigation & decision-making, as well as AI-driven DevOps tools for software development, testing, & deployment

See current & upcoming SEI blog posts on these topics at insights.sei.cmu.edu/blog

Impact on AI-Augmented Software Development



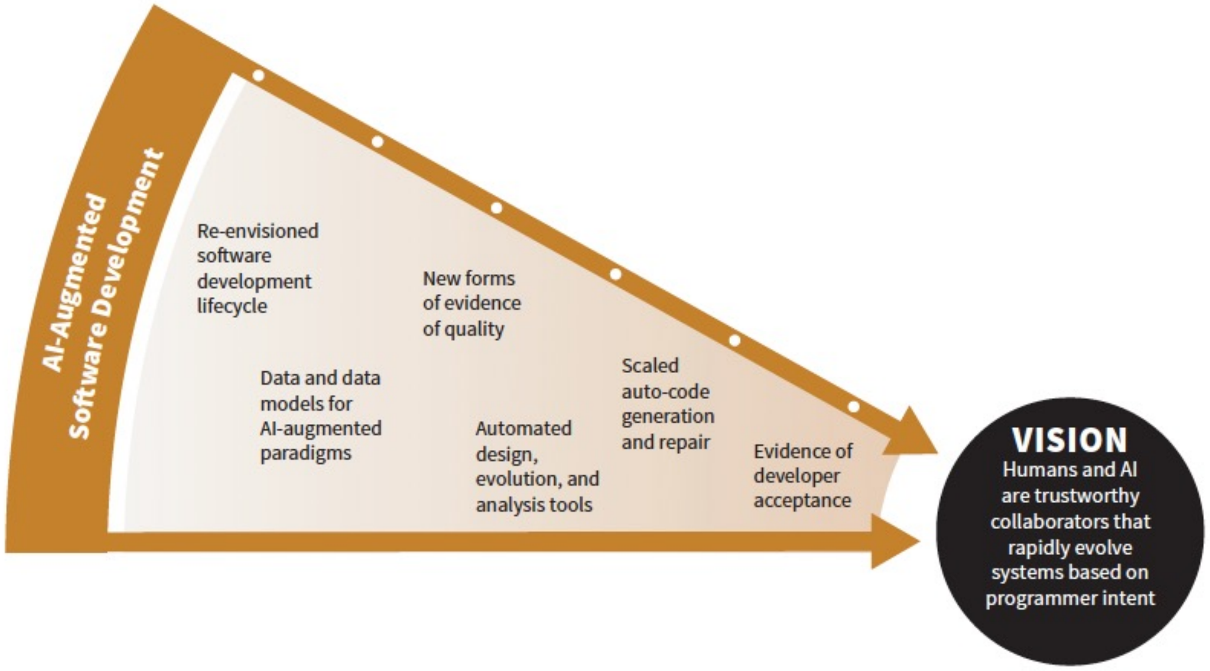
Impact on AI-Augmented Software Development

- We'll start out with a "high-percentage" predication:

See julius-erving-explained-why-he-didnt-attempt-dunks-that-have-good-chances-of-missing

Impact on AI-Augmented Software Development

- We'll start out with a "high-percentage" predication: Generative AI is/will have a transformative impact on the practice of software development



See dev.to/wesen/llms-will-fundamentally-change-software-engineering-3oj8

Impact on AI-Augmented Software Development

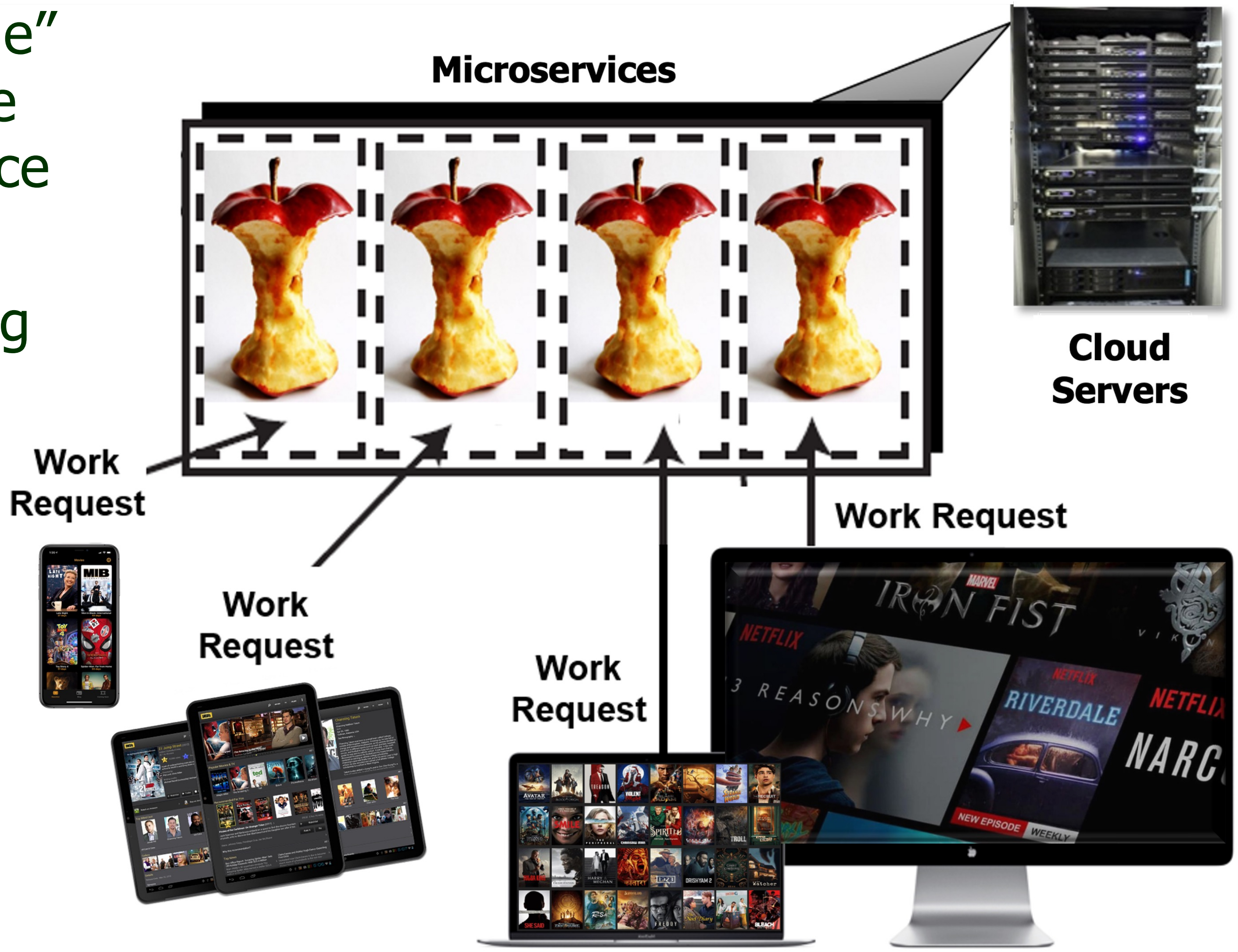
- We'll start out with a "high-percentage" predication: Generative AI is/will have a transformative impact on the practice of software development
  - AI-based tools are increasingly being applied to improve the efficiency & quality of software engineers by reducing their cognitive load

GitHub CoPilot, Amazon CodeWhisperer, Tabnine, Android Studio Bot, etc.

See www.elegantthemes.com/blog/wordpress/best-ai-coding-assistant

Impact on AI-Augmented Software Development

- We'll start out with a "high-percentage" predication: Generative AI is/will have a transformative impact on the practice of software development
  - AI-based tools are increasingly being applied to improve the efficiency & quality of software engineers by reducing their cognitive load



See www.youtube.com/watch?v=tefB7FgYTxE

Impact on AI-Augmented Software Development

- We'll start out with a "high-percentage" predication: Generative AI is/will have a transformative impact on the practice of software development
  - AI-based tools are increasingly being applied to improve the efficiency & quality of software engineers by reducing their cognitive load

Not everyone is equally bullish about the benefits of generative AI for programmers, of course!!!

BLOG@CACM

AI Does Not Help Programmers

By Bertrand Meyer
June 3, 2023
Comments (3)

VIEW AS: SHARE:

Everyone is blown away by the new AI-based assistants. (Myself included: see an earlier article on this blog which, by the way, I would write differently today.) They pass bar exams and write songs. They also produce programs. Starting with Matt Welsh's article in *Communications of the ACM*, many people now pronounce programming dead, most recently *The New York Times*.

I have tried to understand how I could use ChatGPT for programming and, unlike Welsh, found almost nothing. If the idea is to write some sort of program from scratch, well, then yes. I am willing to believe the experiment reported on Twitter of how a beginner using Copilot to beat hands-down a professional programmer for a from-scratch development of a Minimum Viable Product program, from "Figma screens and a set of specs." I have also seen people who know next to nothing about programming get a useful program prototype by just typing in a general specification. I am talking about something else, the kind of use that Welsh touts: a professional programmer using an AI assistant to do a better job. It doesn't work.

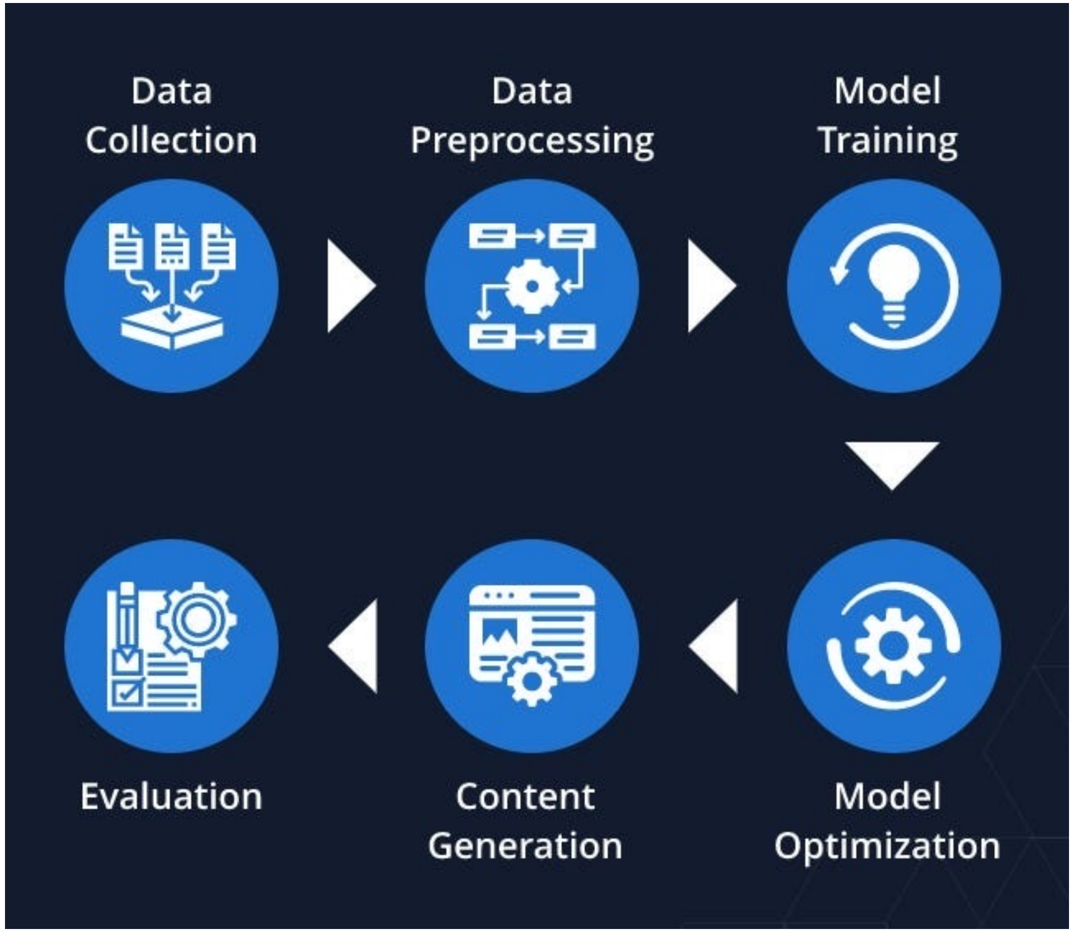
Precautionary observations:

- *Caveat 1*: We are in the early days of the technology and it is easy to mistake teething problems for fundamental limitations. (*PC Magazine*'s initial review of the iPhone: "*it's just a plain lousy phone, and although it makes some exciting advances in handheld Web browsing it is not the Internet in your pocket.*") Still, we have to assess what we have, not what we could get.

See cacm.acm.org/blogs/blog-cacm/273577-ai-does-not-help-programmers/fulltext

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases



See neal-lathia.medium.com/evaluating-llms-trained-on-code-bb2bdab3cb37

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases, e.g.
    - CodeGen is an "autoregressive LLM" for program synthesis trained on The Pile, BigQuery, & BigPython

CODEGEN: AN OPEN LARGE LANGUAGE MODEL FOR CODE WITH MULTI-TURN PROGRAM SYNTHESIS

Erik Nijkamp*, Bo Pang*, Hiroaki Hayashi*, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, Caiming Xiong

Salesforce Research

ABSTRACT

Program synthesis strives to generate a computer program as a solution to a given problem specification, expressed with input-output examples or natural language descriptions. The prevalence of large language models advances the state-of-the-art for program synthesis, though limited training resources and data impede open access to such models. To democratize this, we train and release a family of large language models up to 16.1B parameters, called CODEGEN, on natural language and programming language data, and open source the training library JAXFORMER. We show the utility of the trained model by demonstrating that it is competitive with the previous state-of-the-art on zero-shot Python code generation on HumanEval. We further investigate the multi-step paradigm for program synthesis, where a single program is factorized into multiple prompts specifying subproblems. To this end, we construct an open benchmark, Multi-Turn Programming Benchmark (MTPB), consisting of 115 diverse problem sets that are factorized into multi-turn prompts. Our analysis on MTPB shows that the same intent provided to CODEGEN in multi-turn fashion significantly improves program synthesis over that provided as a single turn. We make the training library JAXFORMER and model checkpoints available as open source contribution: `https://github.com/salesforce/CodeGen`.

1 INTRODUCTION

Creating a program has typically involved a human entering code by hand. The goal of program synthesis is to automate the coding process, and generate a computer program that satisfies the user's specified intent. Some have called it the holy grail of computer science (Manna & Waldinger, 1971; Gulwani et al., 2017). Successful program synthesis would not only improve the productivity of experienced programmers but also make programming accessible to a wider audience.

Two key challenges arise when striving to achieve program synthesis: (1) the intractability of the search space, and (2) the difficulty of properly specifying user intent. To maintain an expressive search space, one needs a large search space, which poses challenges in efficient search. Previous work (Joshi et al., 2002; Panchekha et al., 2015; Cheung et al., 2013) leverages domain-specific language to restrict the search space; however, this limits the applicability of synthesized programs. On the contrary, while being widely applicable, general-purpose programming languages (*e.g.*, C, Python) introduce an even larger search space for possible programs. To navigate through the enormous program space, we formulate the task as language modeling, learning a conditional distribution of the next token given preceding tokens and leverage transformers (Vaswani et al., 2017) and large-scale self-supervised pre-training. This approach has seen success across modalities (Devlin et al., 2019; Lewis et al., 2020; Dosovitskiy et al., 2021). Likewise, prior works have developed pre-trained language models for programming language understanding (Kanade et al., 2020; Feng et al., 2020).

To realize program synthesis successfully, users must employ some means to communicate their intent to the models such as a logical expression (which specifies a logical relation between inputs

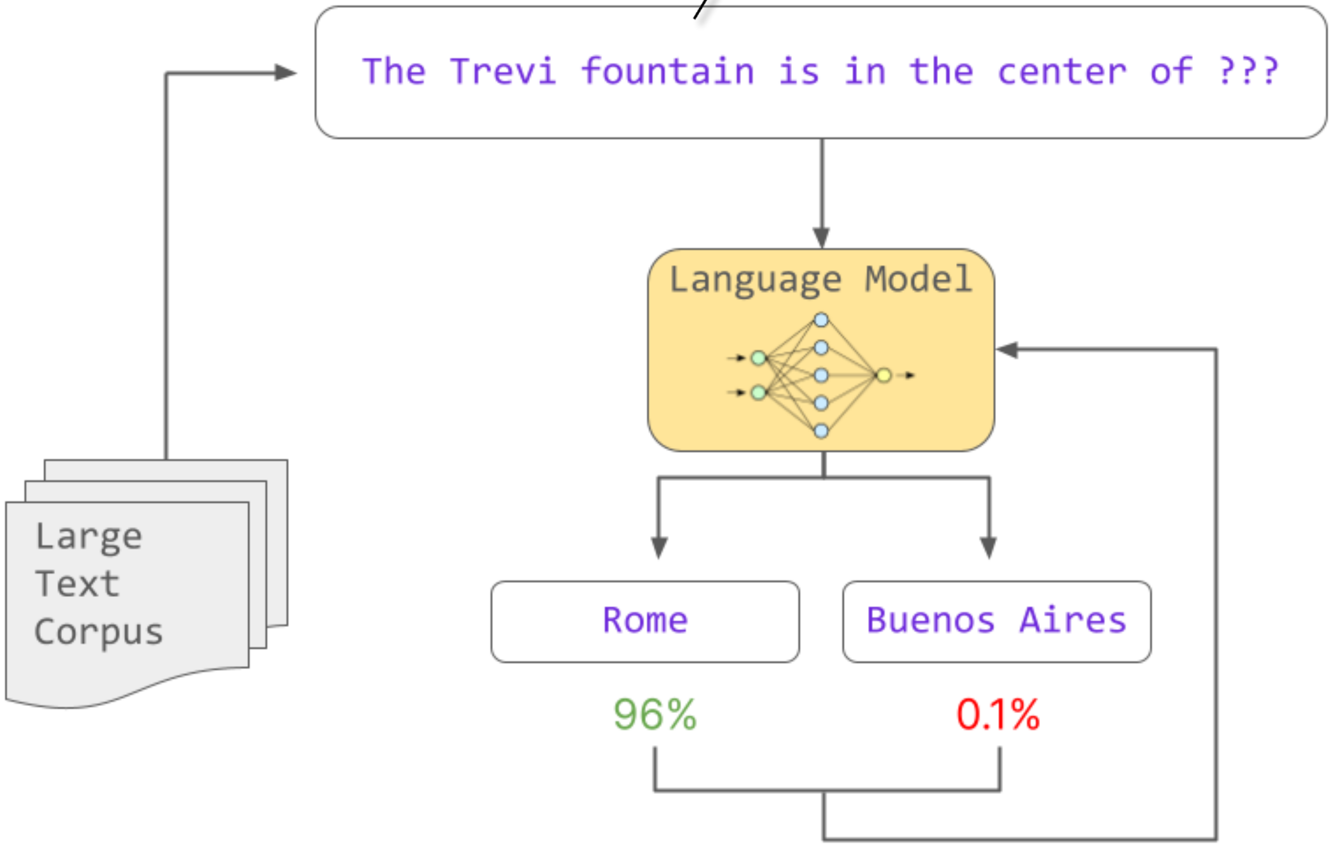
* Equal contribution.
Correspondence to: Erik Nijkamp (erik.nijkamp@salesforce.com), Bo Pang (b.pang@salesforce.com), Hiroaki Hayashi (hiroakihayashi@salesforce.com), Yingbo Zhou (yingbo.zhou@salesforce.com), Caiming Xiong (cxiong@salesforce.com).

See huggingface.co/docs/transformers/model_doc/codegen

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases, e.g.
    - CodeGen is an "autoregressive LLM" for program synthesis trained on The Pile, BigQuery, & BigPython

Autoregressive LLMs generate sequences of text by predicting each token based on the previous tokens in a sequential manner



See www.assemblyai.com/blog/the-full-story-of-large-language-models-and-rlhf

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases, e.g.
    - CodeGen is an "autoregressive LLM" for program synthesis trained on The Pile, BigQuery, & BigPython
      - Its strongest language support is for mainstream languages like Python, JavaScript, Go, & Ruby

CODEGEN: AN OPEN LARGE LANGUAGE MODEL FOR CODE WITH MULTI-TURN PROGRAM SYNTHESIS

Erik Nijkamp*, Bo Pang*, Hiroaki Hayashi*,
Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, Caiming Xiong

Salesforce Research

ABSTRACT

Program synthesis strives to generate a computer program as a solution to a given problem specification, expressed with input-output examples or natural language descriptions. The prevalence of large language models advances the state-of-the-art for program synthesis, though limited training resources and data impede open access to such models. To democratize this, we train and release a family of large language models up to 16.1B parameters, called CODEGEN, on natural language and programming language data, and open source the training library JAXFORMER. We show the utility of the trained model by demonstrating that it is competitive with the previous state-of-the-art on zero-shot Python code generation on HumanEval. We further investigate the multi-step paradigm for program synthesis, where a single program is factorized into multiple prompts specifying subproblems. To this end, we construct an open benchmark, Multi-Turn Programming Benchmark (MTPB), consisting of 115 diverse problem sets that are factorized into multi-turn prompts. Our analysis on MTPB shows that the same intent provided to CODEGEN in multi-turn fashion significantly improves program synthesis over that provided as a single turn. We make the training library JAXFORMER and model checkpoints available as open source contribution: `https://github.com/salesforce/CodeGen`.

1 INTRODUCTION

Creating a program has typically involved a human entering code by hand. The goal of program synthesis is to automate the coding process, and generate a computer program that satisfies the user's specified intent. Some have called it the holy grail of computer science (Manna & Waldinger, 1971; Gulwani et al., 2017). Successful program synthesis would not only improve the productivity of experienced programmers but also make programming accessible to a wider audience.

Two key challenges arise when striving to achieve program synthesis: (1) the intractability of the search space, and (2) the difficulty of properly specifying user intent. To maintain an expressive search space, one needs a large search space, which poses challenges in efficient search. Previous work (Joshi et al., 2002; Panchekha et al., 2015; Cheung et al., 2013) leverages domain-specific language to restrict the search space; however, this limits the applicability of synthesized programs. On the contrary, while being widely applicable, general-purpose programming languages (*e.g.*, C, Python) introduce an even larger search space for possible programs. To navigate through the enormous program space, we formulate the task as language modeling, learning a conditional distribution of the next token given preceding tokens and leverage transformers (Vaswani et al., 2017) and large-scale self-supervised pre-training. This approach has seen success across modalities (Devlin et al., 2019; Lewis et al., 2020; Dosovitskiy et al., 2021). Likewise, prior works have developed pre-trained language models for programming language understanding (Kanade et al., 2020; Feng et al., 2020).

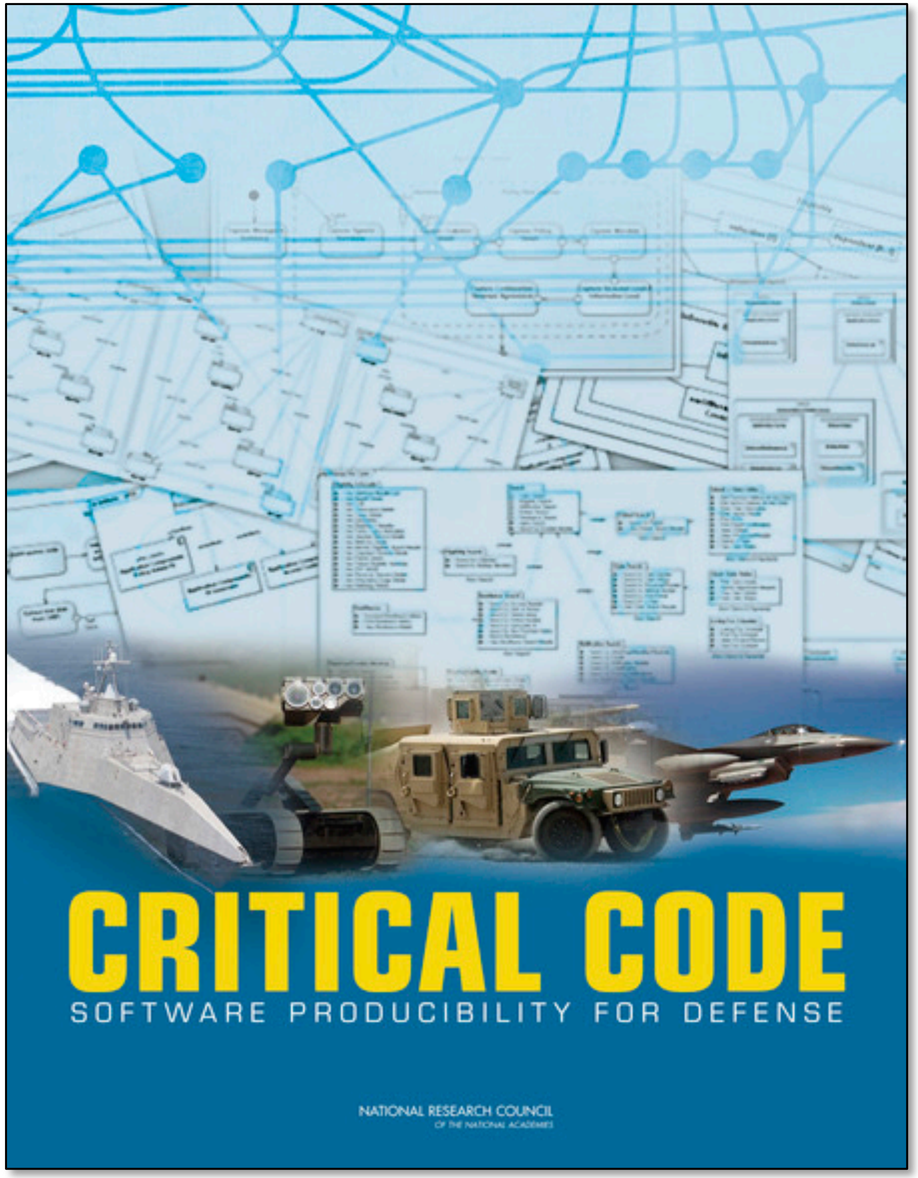
To realize program synthesis successfully, users must employ some means to communicate their intent to the models such as a logical expression (which specifies a logical relation between inputs

* Equal contribution.
Correspondence to: Erik Nijkamp (erik.nijkamp@salesforce.com), Bo Pang (b.pang@salesforce.com), Hiroaki Hayashi (hiroakihayashi@salesforce.com), Yingbo Zhou (yingbo.zhou@salesforce.com), Caiming Xiong (cxiong@salesforce.com).

See huggingface.co/docs/transformers/model_doc/codegen

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases, e.g.
    - CodeGen is an "autoregressive LLM" for program synthesis trained on The Pile, BigQuery, & BigPython
    - Specialized LLMs are useful for communities that have stringent or unconventional quality attributes



See nap.nationalacademies.org/catalog/12979/critical-code-software-producibility-for-defense

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases, e.g.
    - CodeGen is an "autoregressive LLM" for program synthesis trained on The Pile, BigQuery, & BigPython
    - Specialized LLMs are useful for communities that have stringent or unconventional quality attributes, e.g.,
      - Mission- & safety-critical systems

BY ORDER OF THE SECRETARY OF THE AIR FORCE

AIR FORCE MANUAL 91-119

5 JUNE 2012

Safety

SAFETY DESIGN AND EVALUATION CRITERIA FOR NUCLEAR WEAPON SYSTEMS SOFTWARE

COMPLIANCE WITH THIS PUBLICATION IS MANDATORY

ACCESSIBILITY: Publications and forms are available for downloading or ordering on the e-Publishing website at www.e-Publishing.af.mil

RELEASABILITY: There are no releasability restrictions on this publication

OPR: HQ AFSEC/SEWN

Certified by: AF/SE (Maj Gen Feest)

Supersedes: AFMAN 91-119, 1 February 1999

Pages: 30

This manual implements AFPD 91-1, *Nuclear Weapons and System Surety*, and contains the minimum design and evaluation criteria for software requiring nuclear safety certification. It applies to all organizations that design, develop, modify, evaluate, operate or acquire a nuclear weapon system. This publication is consistent with AFPD 13-5, *Air Force Nuclear Enterprise*. This Manual is applicable to Air National Guard and Air Force Reserve units performing nuclear missions. This manual applies to new systems or modified portions of existing systems. Existing certified systems are not required to be modified solely to meet the requirements of this manual. Refer recommended changes and questions about this publication to the Office of Primary Responsibility (OPR) using the AF Form 847, *Recommendation for Change of Publication*; route AF Form 847s from the field through the appropriate (MAJCOM) publications/forms manager. Ensure that all records created as a result of processes prescribed in this publication are maintained in accordance with AFMAN 33-363, *Management of Records*, and disposed of in accordance with the Air Force Records Disposition Schedule (RDS) located at https://www.my.af.mil/afrims/afrims/afrims/rims.cfm. Send recommendations for improvements to Headquarters Air Force Safety Center (AFSEC/SEWN), 9700 G Avenue SE, Kirtland AFB, NM 87117-5670, or email HQAFSCSEWN@kirtland.af.mil

SUMMARY OF CHANGES

This document is substantially revised and shall be completely reviewed. This revision includes substantive changes. It provides nuclear safety design certification and evaluation criteria for software systems, including facilities, used to support, maintain, handle or store nuclear weapons. In addition, organization names were changed to reflect changes since the last

See static.e-publishing.af.mil/production/1/af_se/publication/dafman91-119/dafman91-119_dafgm2023-01.pdf

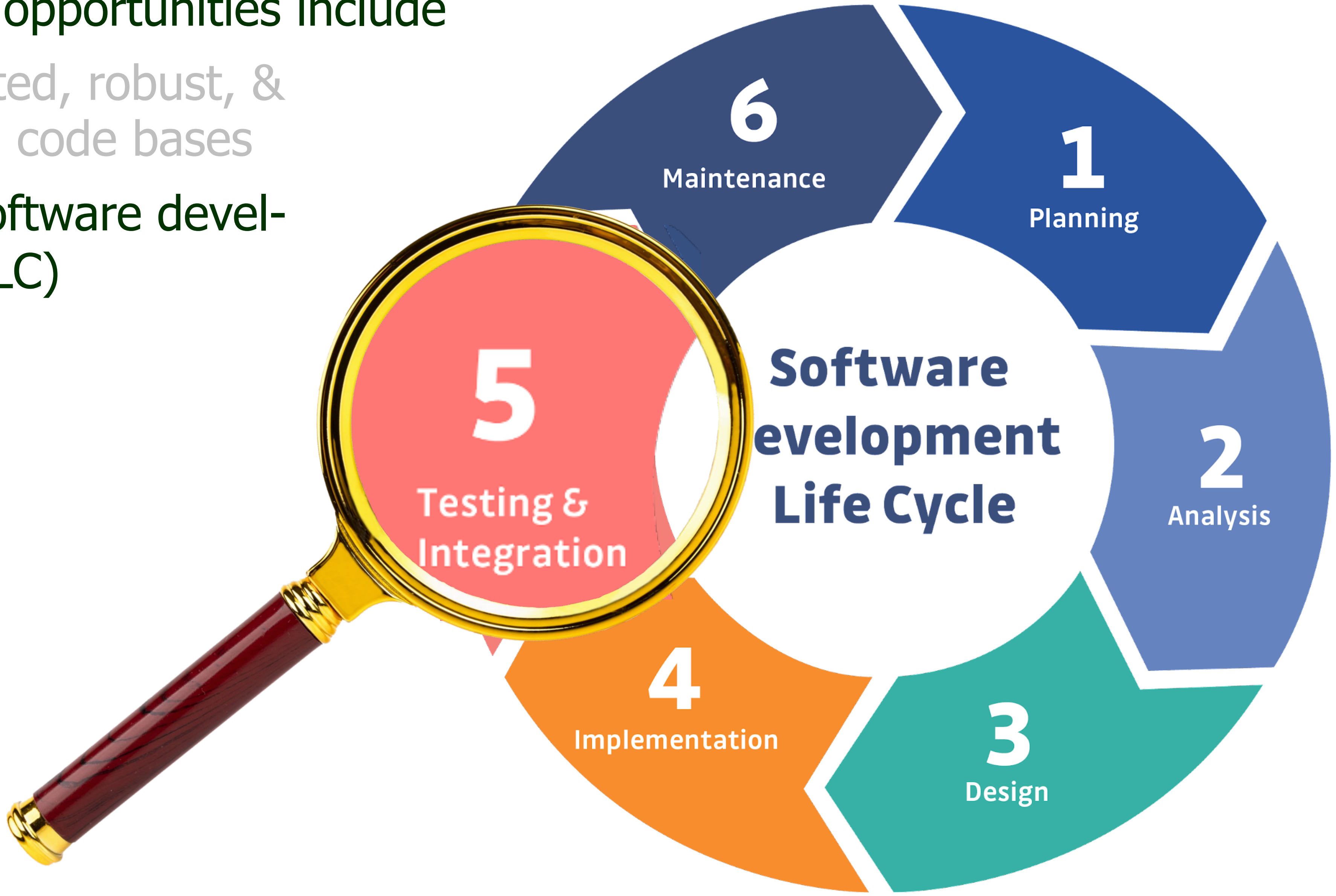
Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases, e.g.
    - CodeGen is an "autoregressive LLM" for program synthesis trained on The Pile, BigQuery, & BigPython
    - Specialized LLMs are useful for communities that have stringent or unconventional quality attributes, e.g.,
      - Mission- & safety-critical systems
      - Legacy systems developed & sustained using non-mainstream programming languages

See nap.nationalacademies.org/read/5463/chapter/3#10

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software development lifecycle (SDLC)



See en.wikipedia.org/wiki/Software_development_process

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software devel-opment lifecycle (SDLC), e.g.
    - Effectively capture/leverage data generated throughout the SDLC

Automatically Detecting Technical Debt Discussions with Machine Learning

ROBERT NORD

APRIL 13, 2020

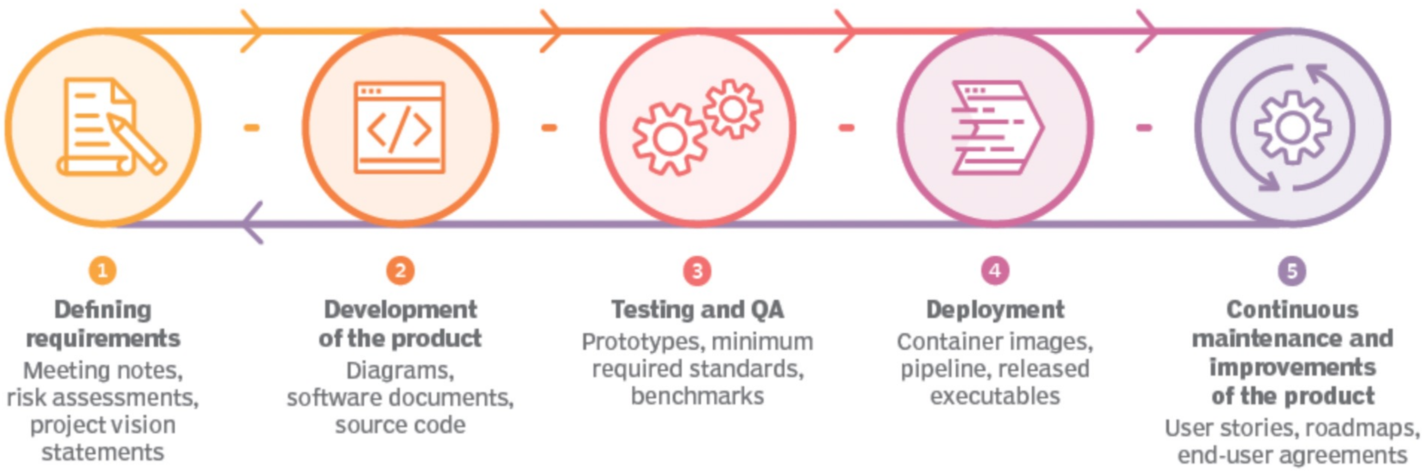
Technical debt (TD) refers to choices made during software development that achieve short-term goals at the expense of long-term quality. Since developers use issue trackers to coordinate task priorities, issue trackers are a natural focal point for discussing TD. In addition, software developers use preset issue types, such as *feature*, *bug*, and *vulnerability*, to differentiate the nature of the task at hand. We have recently started seeing developers explicitly use the phrase "technical debt" or similar terms such as "design debt" or "architectural smells."

Although developers often informally discuss TD, the concept has not yet crystalized into a consistently applied issue type when describing issues in repositories. Application of machine learning to locate technical debt issues can improve our understanding of TD and help develop practices to manage it. In this blog post, which is based on an SEI white paper, we describe the results of a study in which machine learning was used to quantify the prevalence of TD-related issues in issue trackers. Although more work is needed, the study achieved promising results in producing a classifier that automatically determines whether a ticket in an issue tracker relates to TD. Our results suggest the need to designate a new technical debt issue type for technical debt to raise visibility and awareness of TD issues among developers and managers.

See insights.sei.cmu.edu/blog/automatically-detecting-technical-debt-discussions-with-machine-learning

Impact on AI-Augmented Software Development

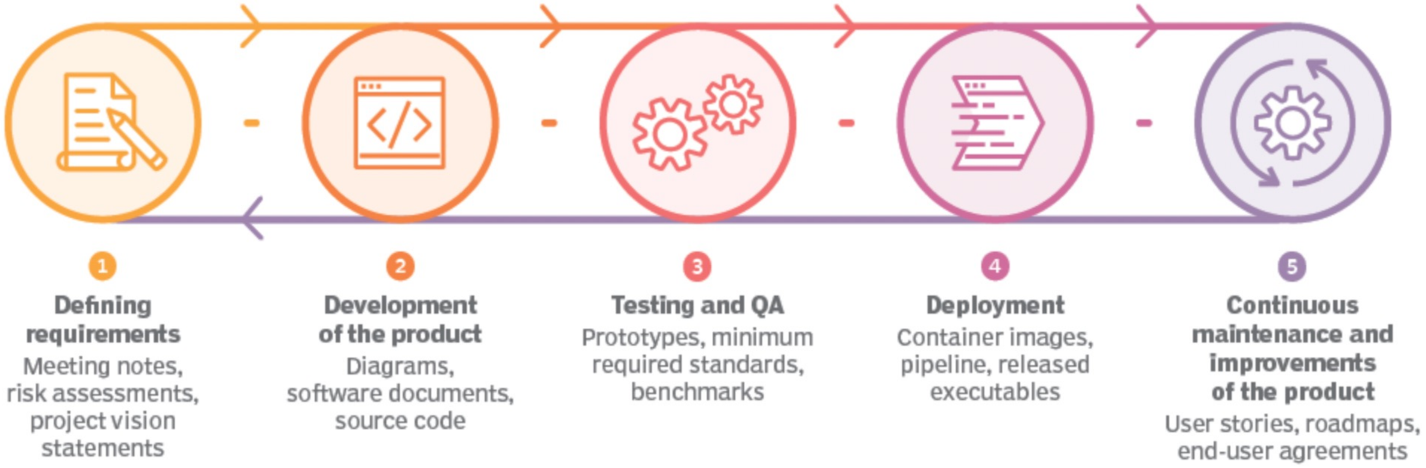
- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software development lifecycle (SDLC), e.g.
    - Effectively capture/leverage data generated throughout the SDLC
      - e.g., many non-code artifacts can be analyzed at scale by AI tools better/ faster/cheaper than by humans alone



See aiperspectives.springeropen.com/articles/10.1186/s42467-020-00005-4

Impact on AI-Augmented Software Development

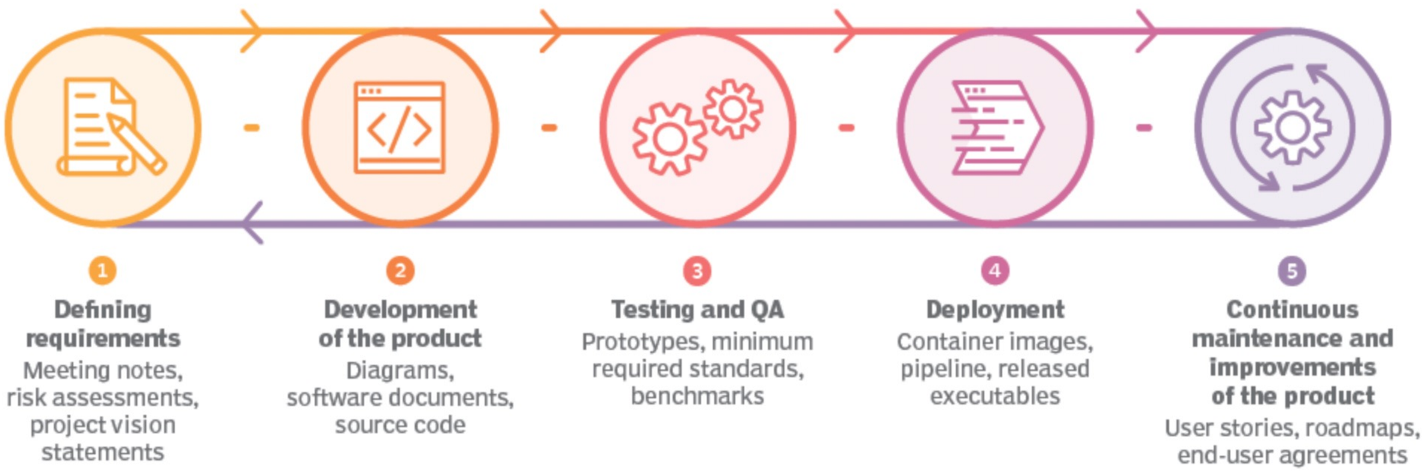
- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software development lifecycle (SDLC), e.g.
    - Effectively capture/leverage data generated throughout the SDLC
      - e.g., many non-code artifacts can be analyzed at scale by AI tools better/faster/cheaper than by humans alone



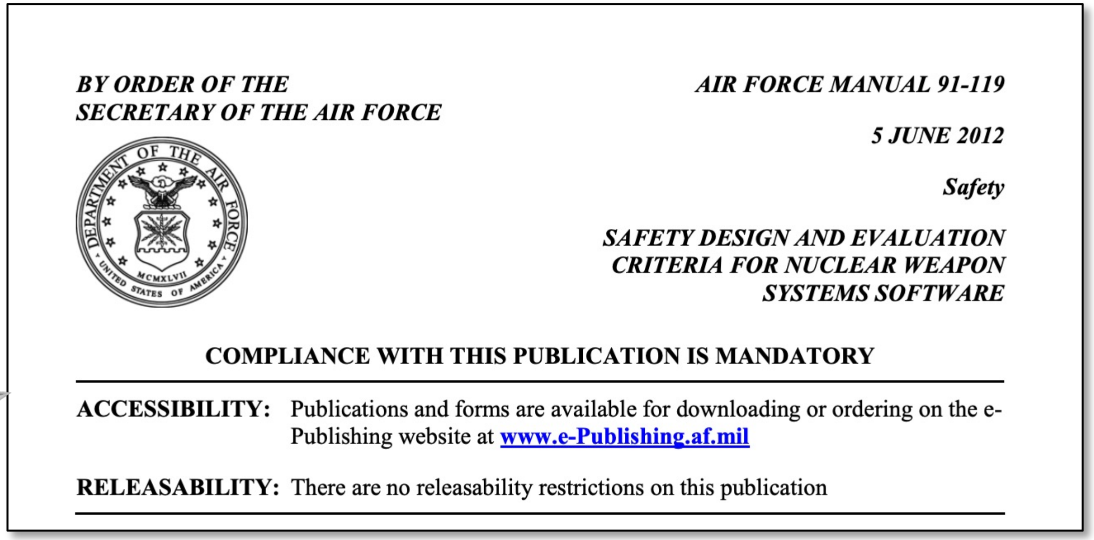
These lifecycle phases are the sweet spot for generative augmented intelligence (AI+) because "utility" is more important than "perfection"

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software development lifecycle (SDLC), e.g.
    - Effectively capture/leverage data generated throughout the SDLC
      - e.g., many non-code artifacts can be analyzed at scale by AI tools better/faster/cheaper than by humans alone



Objectives	Using LLMs
Instructions are clear & complete to enable nuclear surety	Check for inconsistencies • within 91-119 • between 91-119 & other relevant documents



BY ORDER OF THE SECRETARY OF THE AIR FORCE

AIR FORCE MANUAL 91-119

5 JUNE 2012

Safety

SAFETY DESIGN AND EVALUATION CRITERIA FOR NUCLEAR WEAPON SYSTEMS SOFTWARE

COMPLIANCE WITH THIS PUBLICATION IS MANDATORY

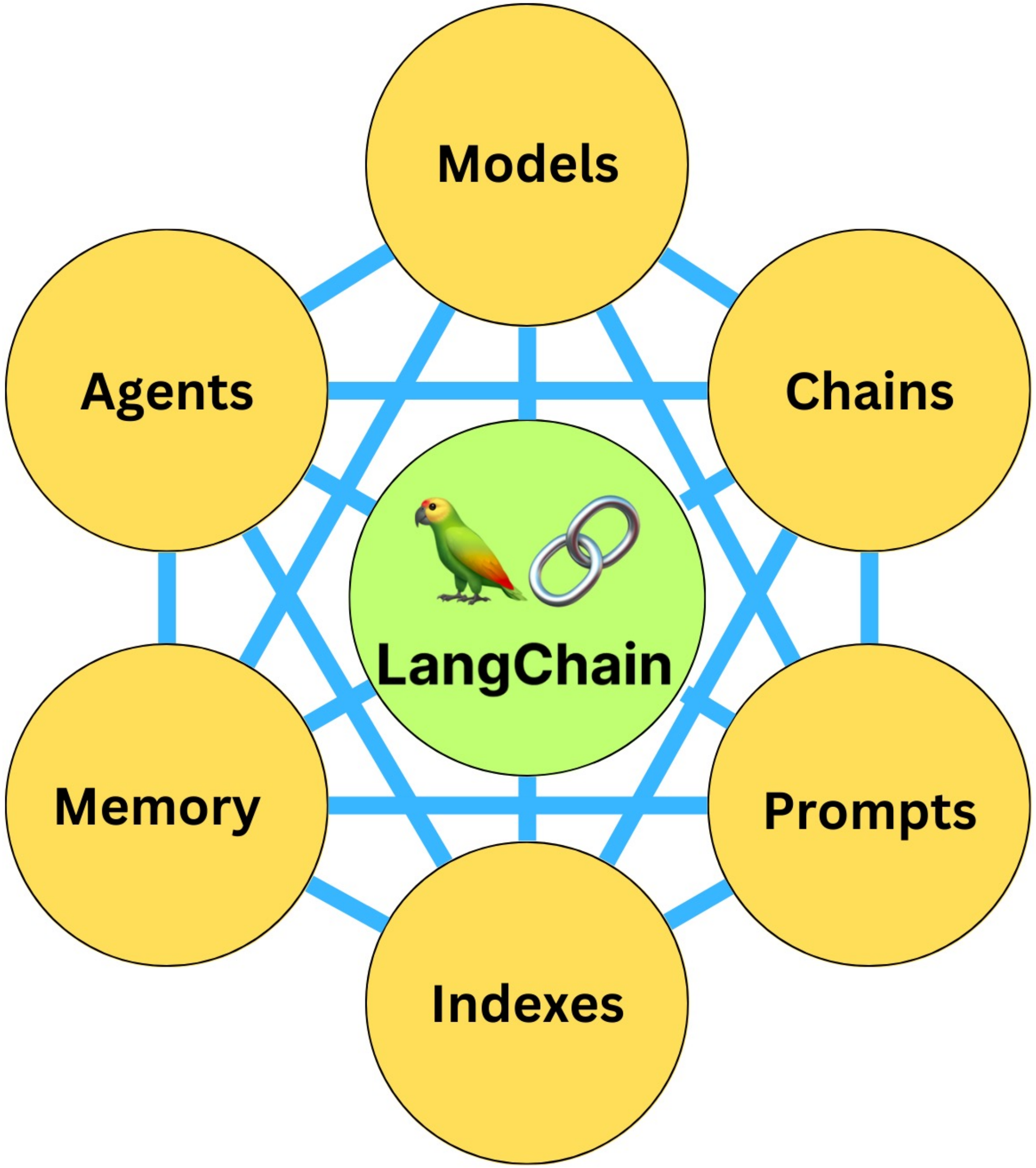
ACCESSIBILITY: Publications and forms are available for downloading or ordering on the e-Publishing website at www.e-Publishing.af.mil

RELEASABILITY: There are no releasability restrictions on this publication

Software Engineering Institute

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software development lifecycle (SDLC), e.g.
    - Effectively capture/leverage data generated throughout the SDLC
    - Increase AI & automation tool support for developers & other stakeholders throughout the SDLC



See docs.langchain.com/docs

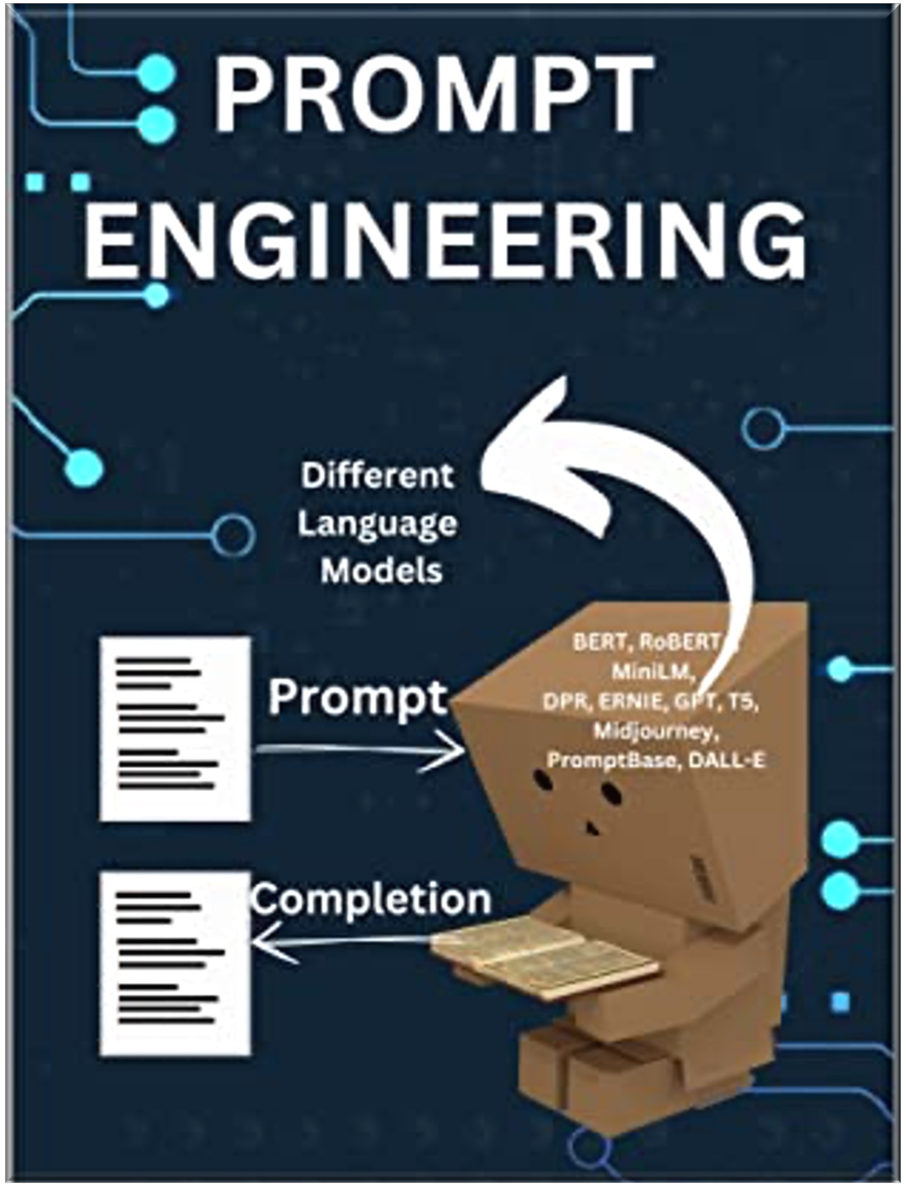
Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software devel-opment lifecycle (SDLC), e.g.
    - Effectively capture/leverage data generated throughout the SDLC
    - Increase AI & automation tool support for developers & other stakeholders throughout the SDLC
      - e.g., check for compliance with relevant policies & standards based on LLM-based static analysis & other static analysis tools

See wiki.sei.cmu.edu/confluence/display/seccode & misra.org.uk/app/uploads/2021/06/MISRA-Compliance-2020.pdf

Impact on AI-Augmented Software Development

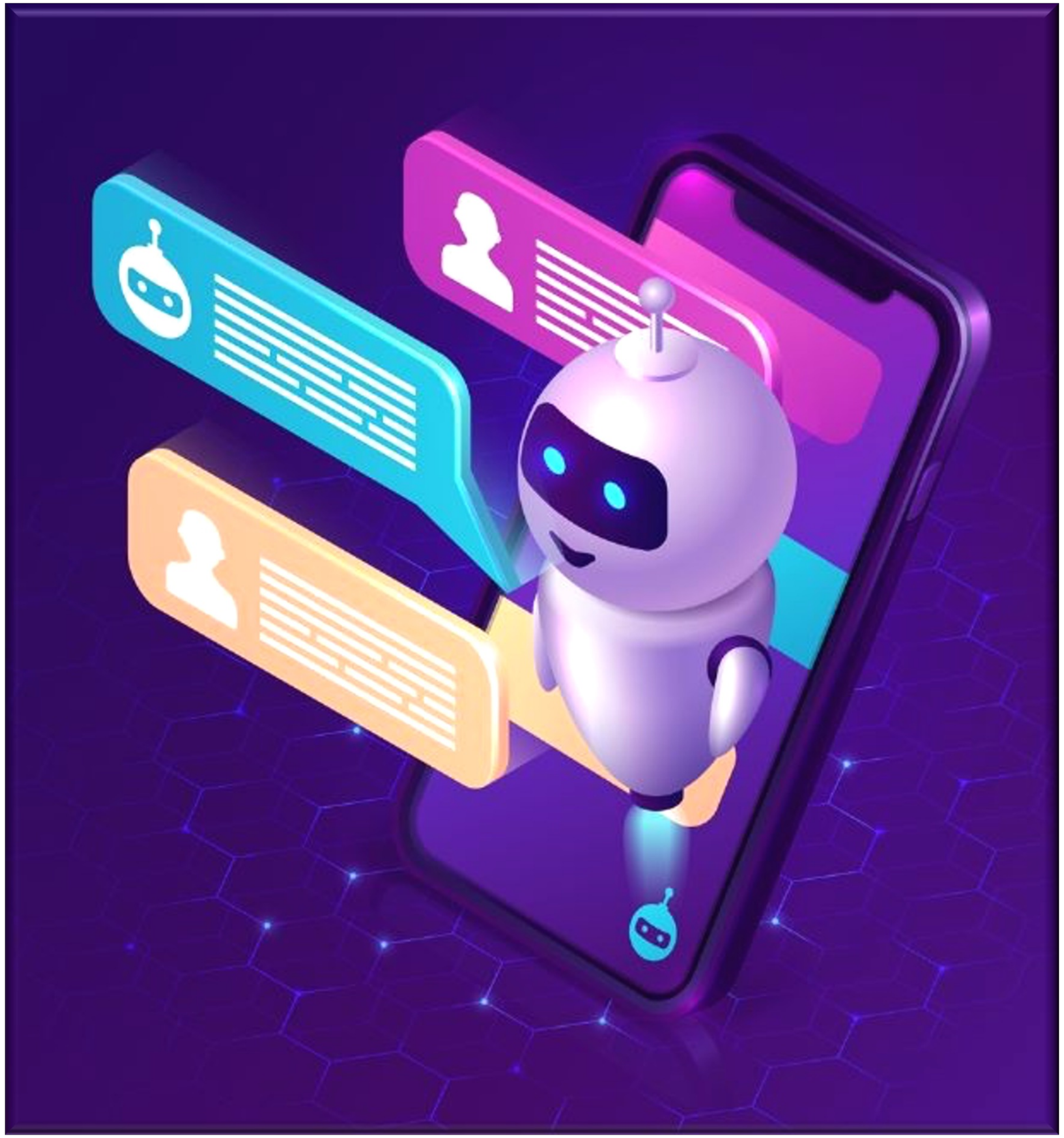
- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software devel-opment lifecycle (SDLC)
  - Formalizing the discipline of "Prompt Engineering"



See en.wikipedia.org/wiki/Prompt_engineering

Impact on AI-Augmented Software Development

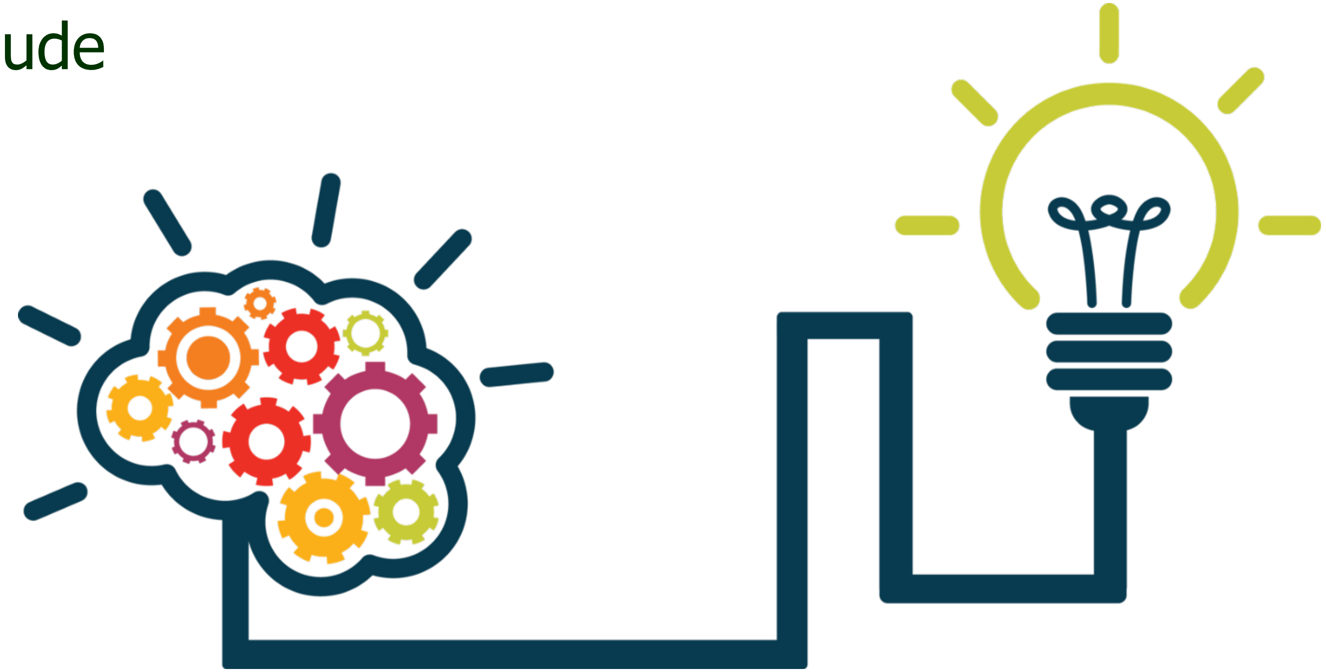
- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software devel-opment lifecycle (SDLC)
  - Formalizing the discipline of "Prompt Engineering", e.g.
    - Learning to "program" using natural language



See en.wikipedia.org/wiki/Prompt_engineering

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software devel-opment lifecycle (SDLC)
  - Formalizing the discipline of "Prompt Engineering", e.g.
    - Learning to "program" using natural language
      - Focus on "problem solving" not traditional computer programming..



See www.youtube.com/watch?v=NrzB6Tb_k2Y&list=PLZ9NgFYEMxp72Zo0yrTNS6utAXxYpqNGl&index=6

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software devel-opment lifecycle (SDLC)
  - Formalizing the discipline of "Prompt Engineering", e.g.
    - Learning to "program" using natural language
    - Codifying "prompt patterns"

Inspired by software patterns, which provide reusable solutions to common problems that occur during software development, providing a template to solve similar issues in various contexts

See www.dre.vanderbilt.edu/~schmidt/POSA

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software devel-opment lifecycle (SDLC)
  - Formalizing the discipline of "Prompt Engineering", e.g.
    - Learning to "program" using natural language
    - Codifying "prompt patterns"

A knowledge transfer method for inter-acting w/large language models (LLMs)

A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT

Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert,
Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt
Department of Computer Science
Vanderbilt University, Tennessee
Nashville, TN, USA
{jules.white, quchen.fu, george.s.hays, michael.sandborn, carlos.olea, henry.gilbert,
ashraf.elnashar, jesse.spencer-smith, douglas.c.schmidt}@vanderbilt.edu

***Abstract*—Prompt engineering is an increasingly important skill set needed to converse effectively with large language models (LLMs), such as ChatGPT. Prompts are instructions given to an LLM to enforce rules, automate processes, and ensure specific qualities (and quantities) of generated output. Prompts are also a form of programming that can customize the outputs and interactions with an LLM.**

This paper describes a catalog of prompt engineering techniques presented in pattern form that have been applied to solve common problems when conversing with LLMs. Prompt patterns are a knowledge transfer method analogous to software patterns since they provide reusable solutions to common problems faced in a particular context, i.e., output generation and interaction when working with LLMs.

This paper provides the following contributions to research on prompt engineering that apply LLMs to automate software development tasks. First, it provides a framework for documenting patterns for structuring prompts to solve a range of problems so that they can be adapted to different domains. Second, it presents a catalog of patterns that have been applied successfully to improve the outputs of LLM conversations. Third, it explains how prompts can be built from multiple patterns and illustrates prompt patterns that benefit from combination with other prompt patterns.

***Index Terms*—large language models, prompt patterns, prompt engineering**

I. Introduction

Conversational large language models (LLMs) [1], such as ChatGPT [2], have generated immense interest in a range of domains for tasks ranging from answering questions on medical licensing exams [3] to generating code snippets. This paper focuses on enhancing the application of LLMs in several domains, such as helping developers code effectively and efficiently with unfamiliar APIs or allowing students to acquire new coding skills and techniques.

LLMs are particularly promising in domains where humans and AI tools work together as trustworthy collaborators to more rapidly and reliably evolve software-reliant systems [4]. For example, LLMs are being integrated directly into software tools, such as Github's Co-Pilot [5]–[7] and included in integrated development environments (IDEs), such as IntelliJ [8] and Visual Studio Code, thereby allowing software teams to access these tools directly from their preferred IDE.

A prompt [9] is a set of instructions provided to an LLM that programs the LLM by customizing it and/or enhancing or refining its capabilities. A prompt can influence subsequent interactions with—and output generated from—an LLM by providing specific rules and guidelines for an LLM conversation with a set of initial rules. In particular, a prompt sets the context for the conversation and tells the LLM what information is important and what the desired output form and content should be.

For example, a prompt could specify that an LLM should only generate code that follows a certain coding style or programming paradigm. Likewise, it could specify that an LLM should flag certain keywords or phrases in a generated document and provide additional information related to those keywords. By introducing these guidelines, prompts facilitate more structured and nuanced outputs to aid a large variety of software engineering tasks in the context of LLMs.

Prompt engineering is the means by which LLMs are programmed via prompts. To demonstrate the power of prompt engineering, we provide the following prompt:

> **Prompt:** "From now on, I would like you to ask me questions to deploy a Python application to AWS. When you have enough information to deploy the application, create a Python script to automate the deployment."

This example prompt causes ChatGPT to begin asking the user questions about their software application. ChatGPT will drive the question-asking process until it reaches a point where it has sufficient information to generate a Python script that automates deployment. This example demonstrates the programming potential of prompts beyond conventional "generate a method that does X" style prompts or "answer this quiz question".

Moreover, prompts can be engineered to program an LLM to accomplish much more than simply dictating the output type or filtering the information provided to the model. With the right prompt, it is possible to create entirely new interaction paradigms, such as having an LLM generate and give a quiz associated with a software engineering concept or tool, or even simulate a Linux terminal window. Moreover, prompts have the potential for self-adaptation, suggesting other prompts to gather additional information or generate related artifacts. These advanced capabilities of prompts highlight the importance of engineering them to provide value beyond simple text or code generation.

Prompt patterns are essential to effective prompt engineering. A key contribution of this paper is the introduction of *prompt patterns* to document successful approaches for

See arxiv.org/abs/2302.11382

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software devel-opment lifecycle (SDLC)
  - Formalizing the discipline of "Prompt Engineering", e.g.
    - Learning to "program" using natural language
    - Codifying "prompt patterns"

TABLE I
CLASSIFYING PROMPT PATTERNS FOR AUTOMATING SOFTWARE ENGINEERING TASKS

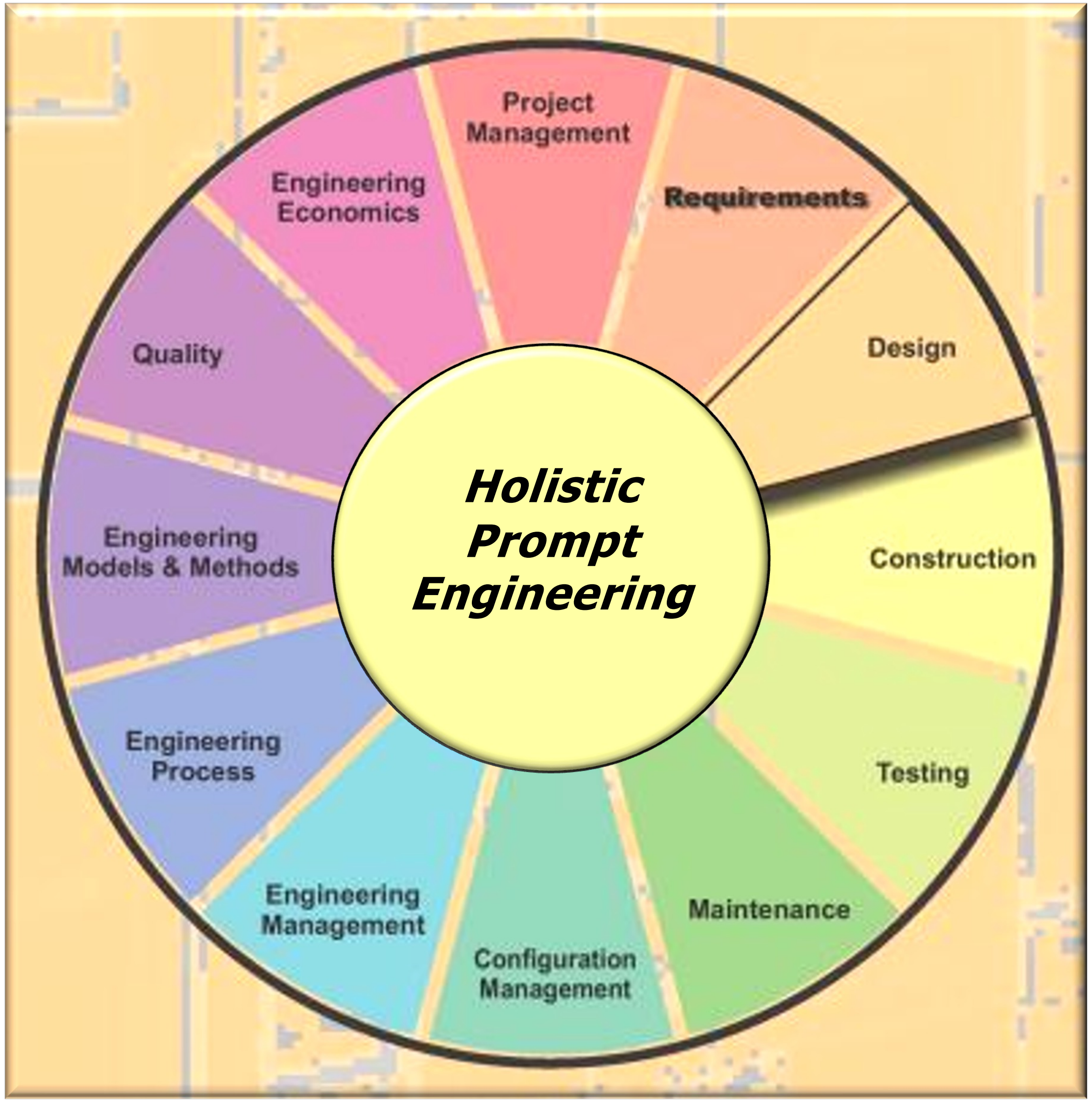
Requirements Elicitation	Requirements Simulator Specification Disambiguation Change Request Simulation
System Design and Simulation	API Generator API Simulator Few-shot Example Generator Domain-Specific Language (DSL) Creation Architectural Possibilities
Code Quality	Code Clustering Intermediate Abstraction Principled Code Hidden Assumptions
Refactoring	Pseudo-code Refactoring Data-guided Refactoring

Define a pattern catalog for automating software engineering tasks that is classified by the types of problems they solve throughout the SDLC

See arxiv.org/abs/2303.07839

Impact on AI-Augmented Software Development

- Key R&D challenges & opportunities include
  - Training LLMs on vetted, robust, & (perhaps) specialized code bases
  - Re-envisioning the software devel-opment lifecycle (SDLC)
  - Formalizing the discipline of "Prompt Engineering", e.g.
    - Learning to "program" using natural language
    - Codifying "prompt patterns"
    - Integrating canonical quality attributes associated with software engineering



See hbr.org/2023/06/ai-prompt-engineering-isnt-the-future

Applying Generative AI to CS Courses at Vanderbilt