

# Software Development Life Cycle for Engineering AI Planning Systems

Ilche Georgievski<sup>a</sup>

Service Computing Department, IAAS, University of Stuttgart, Universitaetsstrasse 38, Stuttgart, Germany

**Keywords:** AI Planning, Software Development Life Cycle, Software Engineering, AI Engineering.

**Abstract:** AI planning is concerned with the automated generation of plans in terms of actions that need to be executed to achieve a given user goal. Considering the central role of this ability in AI and the prominence of AI planning in research and industry, the development of AI planning software and its integration into production architectures are becoming important. However, building and managing AI planning systems is a complex process with its own peculiarities, and requires expertise. On the one hand, significant engineering challenges exist that relate to the design of planning domain models and system architectures, deployment, integration, and system performance. On the other hand, no life cycle or methodology currently exists that encompasses all phases relevant to the development process to ensure AI planning systems have high quality and industrial strength. In this paper, we propose a software development life cycle for engineering AI planning systems. It consists of ten phases, each described in terms of purpose and available tools and approaches for its execution. We also discuss several open research and development challenges pertaining to the life cycle and its phases.


## 1 INTRODUCTION

In 1966, Shakey was the first general-purpose mobile robot with the ability to plan its own actions (Nilsson, 1984). Shakey also marked the inception of Artificial Intelligence (AI) planning, a research and development discipline that deals with the computation of plans of actions for achieving a user goal (Ghahlab et al., 2004). In other words, AI planning is concerned with solving *planning problems* that consist of an initial state of the world, a planning domain model in terms of actions that can change the world, and a goal state of the world that represents the user goal. A plan, which is a course of actions, is solution to a planning problem if it leads from the initial state to the goal. The field of AI planning is now abound in approaches, algorithms and tools for addressing issues of various aspects of the planning process (see Figure 1). In the last few years, there is a growing demand for AI planning technology in real applications, such as space exploration (Chien and Morris, 2014), robotics (Karpas and Magazzeni, 2020), and autonomous driving (Alnazer et al., 2022).

These developments highlight the need for developing AI planning systems and integrating them into existing software architectures. Within AI planning discipline, the engineering and developing AI plan-

ning systems is often portrayed as two things: the development of algorithms and engineering of planning domain models. In practice, these ingredients constitute only a part of what is needed to construct and use operational AI planning systems (Georgievski and Breitenbücher, 2021). Particularly, developing an AI planning system involves many complex activities: choosing the correct underlying planning model, knowledge engineering, dealing with numerous planning functionalities, architecting a design without an established interoperability mechanism for planning software components, choosing suitable planning tools, and collecting and analysing relevant data. As a result, building such AI planning systems is a complex process and demands immense knowledge about the application domain of interest, expertise in AI planning, and proficiency in software engineering.

While most of existing AI planning software artefacts were developed as research projects, it is unclear whether they were developed considering any traditional software development life cycle. Nevertheless, AI planning faces many of the challenges in traditional software development, such as requirements analysis, testing, code review, documentation, etc. AI planning systems, however, are different from traditional software in several ways, and present new challenges that are not accounted for in existing software development life cycles (see (Munassar and Go-

<sup>a</sup>  <https://orcid.org/0000-0001-6745-0063>

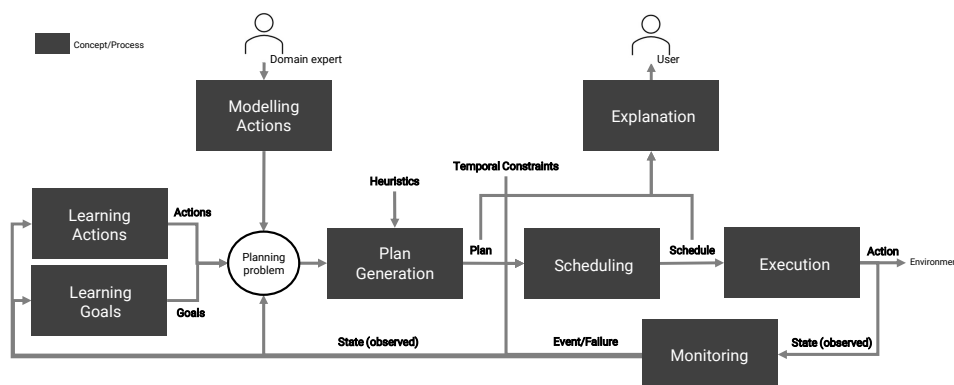


Figure 1: Viewpoint on an AI Planning Process.

vardhan, 2010)), especially in an integrated way. One difference is that the objective in AI planning is to compute and execute plans often optimised with respect to some metric, such as energy cost, instead of only achieving functional requirements. This implies developers would need to continuously improve a planning system (e.g., test newest underlying planning models, update user goals) to achieve the aim (e.g., minimum cost paid for consumed energy). Furthermore, AI planning systems seem to be more complex to manage because their performance depends on data representing the world, planning domain models, tuning algorithms (e.g., selecting or devising suitable heuristics), implicit feedback from the world and explicit user’s feedback, in-depth data analysis (e.g., to design a planning domain model, identify or consider biased/ethical implications of automated planning decisions), distributed computation of a global plan, etc. Finally, AI planning systems are likely developed by people with different expertise, where transferring planning artefacts among them might be challenging. For example, a planning expert might create and transfer a planning domain model to a software engineer for use in the planning system being developed. Any issues that may arise in relation to this might lead to incorrect behaviour (e.g., the system computes plans but they are not valid), which would be hard for the software engineer without planning expertise to identify and address.

As none of the traditional software development life cycles covers the complexity and specificities of AI planning, and no other methodology exists that covers and treats all relevant phases AI planning systems could and should go through, we propose a software development life cycle *for* engineering AI planning systems. The life cycle is an agile and iterative process consisting of ten phases as preliminary conceptualised in (Georgievski, 2023). We describe each phase in detail, including its purpose, the approaches,

and tools currently available to execute it.

Our contributions are manifold. On the one hand, we present a complete software development life cycle aimed at addressing engineering challenges in AI planning. On the other hand, through this lifecycle, we (1) provide a common understanding of how AI planning software is designed, developed, deployed, and refined; this will enable not only cooperation between involved parties (e.g., individuals, groups, mixed AI-human teams) but also facilitate the deployment of AI planning in real applications,<sup>1</sup> (2) render or highlight for the first time some essential aspects of AI planning one needs to consider when building AI planning systems, (3) support the incorporation of quality and effectiveness in the engineering process of AI planning systems, and (4) provide a basis for discussion in AI planning and AI engineering fields and a stepping stone for future research in this direction.

The rest of the paper is organised as follows. Section 2 introduces the proposed life cycle. Section 3 presents an analysis of the lifecycle scope and open challenges. Section 4 describes related works, and Section 5 concludes the paper.

## 2 PROPOSED LIFE CYCLE

To address the question of *what phases a typical AI planning system should go through, how these phases relate to one another, and how they can be conducted*, we present the *Software Development Life Cycle for Engineering AI Planning Systems*, called *PlanXflow*. Figure 2 illustrates the proposed life cycle, which is based on more than ten years of research, development, and teaching experiences in the field. We describe next each of its ten phases.

<sup>1</sup>Only 53% of AI projects move from prototypes to production due to the lack of resources to build and operate

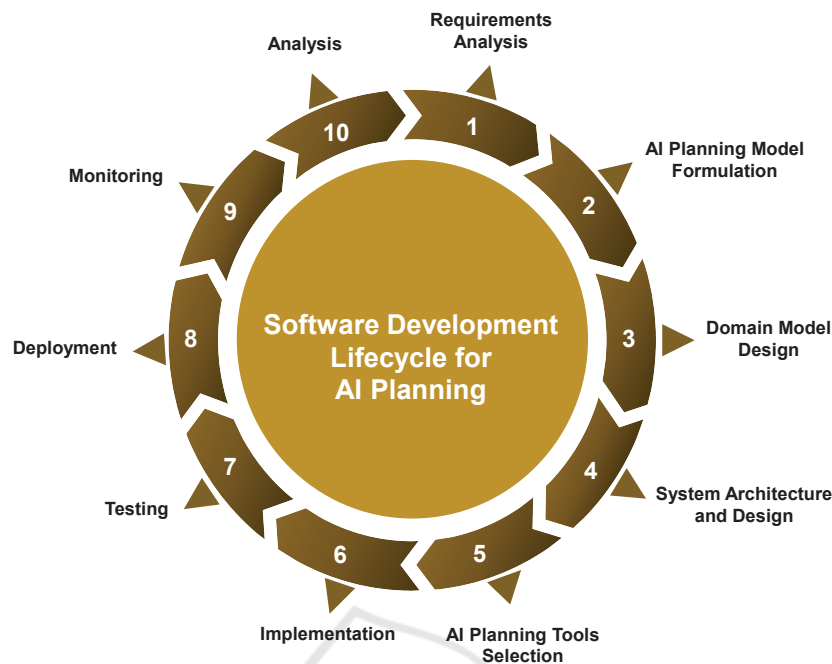


Figure 2: Overview of the Software Development Life Cycle for AI Planning Systems.

## 2.1 Requirements Analysis

The first phase is *Requirements Analysis*, which indicates that the development of AI planning systems must begin with the identification of requirements by the relevant stakeholders (Grady, 2014). In the context of AI planning, one can distinguish functional, non-functional, domain-oriented and user-related requirements. *Functional requirements* define the basic behaviour of the AI planning system, including not only the required functionality but also the kind of planning problems to be addressed and possible restrictions that should be considered in the overall planning and execution process. In particular, planning systems for real-world applications go beyond plan generation only. Such applications often require a wider spectrum of functionalities that range from support for modelling planning domains and problems, to solving planning problems, to executing, validating, and explaining plans and planning decisions, to managing and monitoring the overall planning system. In fact, one can select from and combine at least 19 types of different planning functionalities as reported in (Georgievski, 2022). While all types of planning functionalities can be used to create an advanced AI planning system, not all of them may have the same role in a system. For example, in an online AI planning editor, the Modelling and Parsing

functionalities would be core functionalities, while the Solving functionality would be a supporting one as it is not essential for modelling planning problems but it may make the online editor complete.

*Non-functional requirements* define the quality properties that affect the experience of using AI planning systems, such as performance, scalability, reusability, or maintainability. With the exception of performance of planning systems in terms of time needed to solve planning problems, non-functional requirements are often not considered as important factors in the AI planning literature even though their early detection and addressing into architecture designs can lead to reduction of cost and effort as known from developing software in other contexts.

*Domain-oriented requirements* define the relevant entities an AI planning system can use and objectives it should achieve in the application domain of interest. These requirements cover the goal-oriented requirements defined in the context of AI planning (van Lamsweerde, 2009; Ambreen et al., 2018). Nevertheless, domain-oriented requirements are used to capture knowledge needed for the planning domain and specific planning problems. The acquisition of these requirements is expected to be done in an integrated form, which would enable the translation of these requirements into formal models needed by some planning functionalities (Silva et al., 2020).

*User-related requirements* define requirements relevant for the user. As Asimov's laws express

---

production-ready AI systems (Gartner, Inc., 2020).

that robots must never harm humans in their operation (Asimov et al., 1984), user-related requirements can specify that AI planning systems should not put at risk the safety of humans, disclose privacy-sensitive data, affect the effectiveness and comfort of users in their intentions or when performing their tasks. Finally, the decisions and actions taken by AI planning systems may need to be explained to users (Chakraborti et al., 2020).

The requirements should be attributed with metrics and documented so that the resulting AI planning system can be validated, verified, or evaluated in other lifecycle phases, such as the Analysis phase (see Section 2.10). For example, the quality of planning domain models can be assessed concerning their accuracy, consistency and completeness, the adequacy of the representational language, and the operability of the software component with a Solving functionality concerning the planning domain model (see (McCluskey et al., 2017)).

## 2.2 AI Planning Model Formulation

The *AI Planning Model Formulation* phase is performed to decide and formulate a suitable planning model as a blueprint for the AI planning system. We define a *planning model* as a specific form of planning distinguished by a planning type, world context, and user features. A *planning type* encompasses a set of planning approaches or planning algorithms that share the same underlying structure, assumptions, or qualities. One can distinguish various types of planning. Classical planning is the most basic and restrictive type, where actions play a key role. This planning type is based on several restrictive assumptions that simplify how the world looks like and operates (e.g., the world is static, fully observable, deterministic, and instantaneous). There exist other planning types that relax one or more of those assumptions by adding expressiveness or some algorithmic features. These planning types include numeric planning, temporal planning, probabilistic planning, contingent planning, conformant planning, conditional planning, distributed planning, continual planning, etc. A line of planning structurally different than the aforementioned planning types is represented by hierarchical planning (Georgievski and Aiello, 2015). In addition to actions, hierarchical planning uses tasks and other expressiveness constructs to organise the domain knowledge at different levels of abstraction.

Furthermore, we define a *world context* as a characterisation of an aspect or a situation in the world. A planning model may disregard world contexts to simplify planning problems. A planning model may

tolerate or exploit a world context by anticipating unexpected events in the environment and/or controlling and interacting with the world. So, including a world context in the planning model makes the planning process more complete or realistic. A *user feature* is a something of importance to the user. For example, a domain expert may want to model planning problems or an end-user may need to be provided with an opportunity to explore found plans before their execution. Note that a planning type might exist that covers some world context or user feature. Analysis of their mutual inclusiveness is out of the scope of this work.

The kind of planning model dictates different needs and considerations when developing planning domains and when designing and realising the overall AI planning system. Consequently, the formulation of an adequate planning model is a critical phase. The formulation should be based on all requirements from the previous phase, particularly the functional and domain-oriented requirements. Formulating an adequate planning model is a complex task and may require expertise and experience in AI planning. It can also be time-consuming.

## 2.3 Domain Model Design

As AI planning is a knowledge-based AI approach, the design of systems requires engineering of application domain knowledge in a precise and correct form so that the systems can produce and execute valid plans. In AI planning, one sees knowledge engineering as a modelling process (Studer et al., 1998), which involves constructing a *domain model* that abstractly and conceptually describes the knowledge of the application area. A *planning domain model* is a formal representation of such a domain model.

The *Domain Model Design* phase is dedicated to the creation of a planning domain model. In general, the design process starts by deriving domain information from the requirements collected in the first phase, specifically, from the domain-oriented requirements. On the one hand, the domain information includes object classes, properties, relations, actions, and tasks, and on the other hand, assumptions and features that are essential for the representation of the planning domain model (e.g., propositional logic at minimum). The domain information is then conceptualised and formalised using a selected planning language compatible with the planning model formulated in the previous phase. Examples of planning languages include Planning Domain Definition Language (PDDL), Hierarchical Domain Definition Language (HDDL) (Höller et al., 2020), SHOP2 (Nau et al., 2003), etc. When encoded in such a language,

the planning domain model can be fed in into a software component with a Parsing and Solving functionalities, assuming the component supports the chosen planning syntax and all expressiveness constructs used in the planning domain model.

Domain modelling is a tedious and error-prone task without a clear and established methodology for approaching it and engineering the knowledge in the required form. However, some studies analyse domain modelling issues and propose separate design processes one could follow, e.g., (Vaquero et al., 2011a; Silva et al., 2020). Some modelling tools can smooth down the difficulty of knowledge engineering and support the domain modelling, e.g., itSimple (Vaquero et al., 2013).

## 2.4 System Architecture and Design

Addressing a real-world planning problem will likely require combining several planning functionalities into a single planning system that can solve the problem and satisfy stakeholders' requirements. That is, using the functional requirements, planning model and planning domain model defined in the previous phases, the *System Architecture and Design* phase is concerned with conceptualising an abstract architecture that consists of suitable planning components and relationships between them. Doing this enables not only analysing the planning system's behaviour but also making early design decisions that can impact the development and deployment of the resulting planning system. The planning model is essential here as it directs the conditions for interactions between the planning components to ensure their compatibility and correctness of the entire process. That is, the interaction constraints represent the allowed connections among the constituting components. If necessary and possible, the system architecture can be refined considering the specificities of the involved planning tools, such as data structures or supported features (e.g., some planning tools may not support logical negation even though logical negation is part of PDDL). In any case, the resulting system architecture design should provide sufficient information for the selection and/or implementation of required components in the next phases.

Consider that existing planning tools are mostly developed in isolation without considering communication and interoperability issues, i.e., the tools are designed, developed, and tested per type of planning functionality. For example, domain modellers are designed and developed independently from planners, which the domain modellers would eventually depend upon. Placing needed planning functionalities (i.e.,

their components) into context is investigated on several occasions where only a limited number of coarse-grained planning components are organised into system architectures. For example, CPEF (Myers, 1999), PELEA (Alc et al., 2012), and SOA-PE (Vulgarakis Feljan et al., 2015) provide planning architectures that consist of components for solving planning problems, executing plans, monitoring the execution, and replanning hardwired to operate in a predefined order. In CPEF, a central component manages the operation of the entire planning system.

To design and develop new AI planning systems that are robust, flexible and scalable, the different planning functionalities could be designed and realised as services (Georgievski, 2022), and their coordination and data exchange can be accomplished using workflows (Georgievski and Breitenbücher, 2021), which represent a proven and effective approach for orchestration in many applications (Leymann and Roller, 1997).

## 2.5 AI Planning Tools Selection

When we have a planning problem we want to solve, we usually look for an off-the-shelf tool. In this context and in line with the reusability intention within the AI planning community in terms of providing domain-independent tools and the reusability goal of AI engineering (Georgievski and Breitenbücher, 2021; Bosch et al., 2021), we should explore and select if available existing planning tools before entering the phase to implement the required planning components. In the *AI Planning Tools Selection* phase, existing planning tools (e.g., code, tools, services) are searched to find and select appropriate implementations for the needed planning functionalities.

Research in AI planning has traditionally concentrated on developing techniques and algorithms that are continuously enhanced with delicate features. The result is a large corpus of planning tools, most of which offer a Solving functionality and are called *planners*. For example, one can count 235 planners as participants in International Planning Competition from 2006 to 2020. Other planning tools focus on modelling planning problems, learning planning domains, and validating plans. In addition, the planning tools are diverse in terms of supported operating systems, dependencies, programming languages, and supported planning languages. Suppose one analyses the ten most cited planning tools. One can observe that many of them are restricted to running on a Linux environment, they have at least three dependencies without support for their management, they are implemented in various programming languages with



C, C++, Java, and Common Lisp being the most common ones, and most of these planners support some version of PDDL.

The presented discussion indicates it is challenging to keep track of all the planning tools and know their characteristics and the features they support. Some initial work has been carried out in collecting and categorising planning tools. For example, Georgievski and Aiello characterise several hierarchical planners (Georgievski and Aiello, 2015), while Planning.Wiki provides a list of planners accompanied by a short description and references (Green et al., 2019). In any case, those interested in integrating a planning tool into a planning system have much work to do when looking for a suitable planning tool.

## 2.6 Implementation

The *Implementation* phase is devoted to implementing the planning system based on the system architecture design. The implementation includes the development of components for planning functionalities for which no existing tool can be found, and the development of communication mechanisms, depending on the type of communication required between the planning components. This phase may also be concerned with the modification of planning domain models as a consequence of the capabilities of selected planning tools (e.g., a planning domain model may need to be updated to account for the inability of a planner to deal with negative propositions). For each planning component to be implemented, the classical software development lifecycle can be entered at this point.

## 2.7 Testing

In the *Testing* phase, the AI planning system is tested to validate and verify its behaviour concerning the requirements, quality attributes, and corresponding metrics specified in the first phase. As for other software, two types of testing should be performed: isolated tests and integration tests (Wu et al., 2003). All planning artefacts, such as planning components, the planning domain model, problem instances, and workflow models, should be tested in isolation. To our knowledge, there is only an established strategy for testing planning domain models. In particular, a planning domain model is typically debugged and validated by testing a selected planner to solve a particular problem instance using the planning domain model. The quality of a planning domain model is conventionally assessed concerning adequacy, while other metrics include accuracy and operability.

The AI planning research community has established a practice for testing planners in occasional competitions for the best planner in terms of planning time and plan quality on a set of benchmark planning domains. While some benchmark planning domains are inspired from real-world applications, most of them represent synthetic knowledge suitable for testing the speed of computation and possibly for debugging purposes. Georgievski and Aiello (2016) present a strategy for testing planners with the purpose of understanding how well the planners meet the application requirements. The strategy consists of specifying a testing configuration, algorithmic configuration, knowledge base about relevant planning problems, computational factors, and scalability considerations.

Integration tests are needed to verify that all planning artefacts can work correctly together. To the best of our knowledge, a holistic testing approach for planning systems has not been established yet.

## 2.8 Deployment

The *Deployment* phase is used to make the AI planning system ready and available for use. A centralised AI planning system can be deployed to an environment with enough processing power. Here, the constituent planning components can be deployed manually. However, the manual process of installation, configuration, and deployment of existing planning tools not only requires technical proficiency given their heterogeneity but also is time consuming and error prone given the obscurity of setup instructions (Georgievski and Breitenbücher, 2021). Automated deployment represents a better fit for centralised AI planning systems. Unfortunately, not much information on this topic can be found in the AI planning literature.

Moving an AI planning system with distributed planning to production can be even more challenging, mainly for two of its processes, plan generation and plan execution. Distributed planning and execution require distributed planning components to have common objectives and representations, exchange information about their current plans, and plans iteratively refined and revised until they fit together. There are many possible deployment environments (e.g., cloud, fog and edge), however, there is no standard way to deploy the planning elements (e.g., which parts of planning domain models should be deployed where) to these diverse environments and guarantee successful and correct operation of the AI planning system.

The successful operation of AI planning systems depend on not only on how well their software com-

ponents and planning domain models are engineered, but also on the reliability of the entire process starting from collecting data from the world environment, transforming it to correct planning problem instances, to computing and executing valid plans back in the world environment. All this requires care in reproducing the software environments, the entire process with relevant parameters, etc. used in development.

## 2.9 Monitoring

In the *Monitoring* phase, the planning system and execution environment are monitored to understand the system's behaviour and its constituent components, especially the cases of problems and unexpected behaviour. Monitoring entails collecting data provenance. Generally, provenance is the documentation of the information that describes the origin and production process of an object or a piece of data (Herschel et al., 2017). The information typically includes metadata about entities, data, processes, activities, and people involved in the production process. The collection and processing of provenance provide several benefits, such as improving the understanding of the process and the result, quality assessment and quality improvement, ensuring reproducibility, and technical requirements, such as runtime performance and scalability. Of interest in our treatment is data provenance, which is defined as the information about individual data items and operations involved with those data items (Herschel et al., 2017).

In the context of AI planning, the literature has mostly focused on studying provenance of plans, particularly plan rationales, which deal with “why a plan is the way it is” and “the reason as to why the planning decisions were taken” (Polyak and Tate, 1998), and plan evaluation rationales, which deal with “why a certain plan element does or does not satisfy a criterion” or “why a plan was classified into a specific quality level” (Vaquero et al., 2011b). However, data provenance is crucial not only for understanding the life cycle of plans, but also the correctness, accuracy, and other quality attributes of other planning artefacts (e.g., planning domain model, problem instances), improving the planning process, establishing causality and dependency, establishing responsibility (e.g., who is responsible for which data), and explainability. Also, data provenance can be used to support the selection of planning tools and to improve the performance of the overall planning systems.

Based on this discussion and from our own experience, we present a preliminary collection of data provenance types that can be collected and analysed in AI planning systems. We organise data provenance

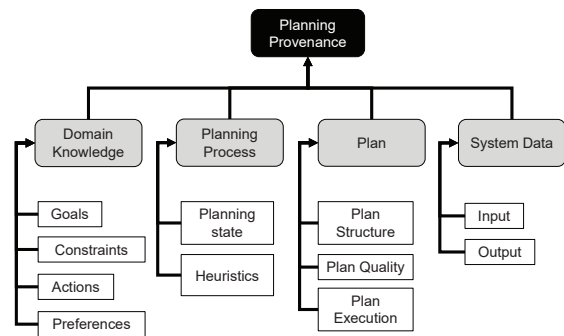


Figure 3: Preliminary Types of Data Provenance in AI Planning.

in AI planning into four categories as shown in Figure 3. The category of *Domain Knowledge* indicates that information about the planning domain should be collected, such as identification of new constraints or preferences that were initially not obvious, or identification of additional requirements for goals or actions. The collection and analysis of this category of data provenance would enable refining the planning domain, including both the domain model and problem instances. The category of *Planning Process* focuses on data provenance about the state of world used in the planning process and heuristics used to guide the plan generation. The *Plan* category includes provenance about the structure and quality of plans, which relate to the plan rationales and plan evaluation rationales, respectively, and provenance about plan execution, which is especially important as sensors, software components, unexpected events, people, and execution of a previous plan can influence the execution of the current plan (Canal et al., 2020). Finally, the category of *System Data* refers to the input data and output data that are needed for the operation of the resulting planning system.

## 2.10 Analysis

The *Analysis* phases is dedicated to analysing data provenance collected in the previous phase. The analysis requires capabilities to identify problems, ask questions and generate relevant insights with the objectives of improving all aspects of the planning system (e.g., domain knowledge, planning process), enable traceability, reproducibility, and explainability.

When this phase is concluded, the next iteration of PlanXflow can be started where actions are taken according to the generated insights in the analysis phase, e.g., refine the planning domain model or integrate an alternative component with a Solving functionality.

### 3 DISCUSSION

PlanXflow represents a general process for designing, developing, and deploying AI planning systems. As such, adjusting some of the lifecycle phases to the specific requirements of some AI planning systems might be necessary. For example, the Domain Model Design phase can be adjusted to include the verification and validation steps of the planning domain model (Silva et al., 2020). Using our structured approach for engineering AI planning systems may conflict with a more loose approach adopted in the research. However, the proposed life cycle has a certain degree of flexibility. Developers can work with selected phases and adjust the order of those phases as needed. Some phases are part of the general life cycle because of the limited understanding of AI planning by non-experts. For example, the AI Planning Tools Selection phase might not be needed if a planning expert is available to point out the right tools for addressing the problem at hand. Furthermore, we suggest using workflows to orchestrate non-trivial planning applications in the System Architecture and Design phase. However, the life cycle can still be used even if workflows might not be needed for a planning system.

Some phases can drastically change due to new advancements of AI planning or other technology. For example, the Requirements Analysis phase could be adapted to include AI-driven requirements (Bosch et al., 2018), the AI Planning Model Formulation phase can be adapted or completely discarded if some method appears to be good at providing relevant recommendations, or the System Architecture and Design phase may need to be adjusted if a toolbox of standardised planning services becomes available.

The presented software development life cycle is intended to provide the basis for discussions among practitioners and researchers of AI planning, and to potentially enable future research in this direction. For example, there are several open challenges that can be addressed to support individual phases and improve the current life cycle. One such challenge is the formulation of suitable planning models, which could be addressed by developing a decision-support system. Another challenge is the accessibility and selection of suitable planning tools. One option to address this would be to have a database of available planning tools with details relevant to development, a toolbox, or even a recommender system that will assist users in finding which tool is suitable for addressing which part of the planning problem, thus facilitating the development of planning systems. In this context, portfolio-based approaches, especially online

ones, could be useful as they learn to suggest the right planner for a given task (Ma et al., 2020). Furthermore, the design of a holistic strategy for testing planning systems and the development of isolated testing strategies for individual planning artefacts are open research challenges. For deploying AI planning systems, we need to investigate appropriate deployment models for both centralised and distributed planning where all required planning components and deployment information are described in a standard manner, providing for reusability and maintainability.

One could upgrade the life cycle by defining separate life cycles for selected planning artefacts and interweaving them in the corresponding phases. For example, a separate life cycle for designing domain models can be created and integrated in the Domain Model Design phase. Finally, a further analysis of data provenance in AI planning might be necessary to refine even more the hierarchy presented in Figure 3, and new approaches are needed to collect and analyse data provenance for all categories.

### 4 RELATED WORK

To the best of our knowledge, there is no work that studies the software development life cycle for constructing AI planning systems. There are some studies, guidelines and established practices that are related to some aspect of some of our individual phases. For example, Silva et al. (2020) focus on the process of designing planning domain models for which they present several phases. This process is compatible with our phase of Domain Model Design and can even serve as a basis for creating a separate Domain Model Design life cycle. Pellier and Fiorino (2018) provide guidelines on how to setup and build PDDL4J on a host platform accompanied with a set of planning problems encoded in PDDL to test the planners available in PDDL4J, and on how to integrate the available planners in third-party applications. Muise et al. (2022) provide guidelines for using PLANUTILS, a library for setting up Linux environments for developing, executing, and testing planners. The guidelines explain how to use the planners available in the library and how to add a new package to the library, for example, if one wished to add a new planner.

Outside the AI planning field but within AI, one can observe the appearance of a couple of new software development life cycles created for specific types of systems. Zaharia et al. (2018) analyse challenges in developing machine learning based systems and describe a platform that can be used to structure and streamline the machine learning development



lifecycle. Olszewska (2019) describes challenges in engineering intelligent vision systems and proposes an adapted software development life cycle with the aim to support developers create quality intelligent vision systems of the new generation. Outside AI, Weder et al. (2022) discuss the challenges quantum computing introduces when engineering quantum applications and proposes an adapted software development life cycle aim at developing and executing such applications.

## 5 CONCLUSION

AI planning originated in the mid-1960s, and since then, many approaches have been proposed mainly around two concerns, namely knowledge representation, and reasoning. Those approaches have demonstrated the excellence of AI planning by prototyping planning tools and evaluating them in experiments. The prominence of AI planning is growing by being integrated into real-world applications, highlighting the need to build AI planning systems for prototyping and experiments, and industrial deployments. Despite the fruitfulness of the discipline with algorithms and planning tools, especially planners, a complete and integrated overview of the phases needed to design, develop, and deploy planning systems is missing. Moreover, a common understanding of relevant phases is needed, which would not only serve interdisciplinary development teams and non-experts but would also enable discussion within the AI planning community and the emerging AI engineering community and possibly foster new research. Therefore, we introduced PlanXflow, a software development life cycle for engineering AI planning systems, which consists of ten phases that may happen during the design, development, deployment, and use of planning systems. For each phase, we described the objective, the approaches, and the tools available for its execution. In addition, we explained and highlighted the need for formulating adequate planning models, selecting suitable planning tools, production deployment, and activities typically taken for granted. We also motivated and highlighted the need for provenance in AI planning to collect and analyse data in relevant phases. We also pointed out several open challenges, which along with the application of our life cycle on a case study and comparison with other lifecycle models, are of interest for future work.

## ACKNOWLEDGEMENTS

I am grateful to Ebaa Alnazer for the comments an early draft of this paper. I thank Marco Aiello for the insightful discussions.

## REFERENCES

- Alc, V., Prior, D., Onaindia, E., Borrajo, D., Fdez-Olivares, J., and Quintero, E. (2012). PELEA: a Domain-Independent Architecture for Planning, Execution and Learning. In *Scheduling and Planning Applications workshop (SPARK)*.
- Alnazer, E., Georgievski, I., Prakash, N., and Aiello, M. (2022). A Role for HTN Planning in Increasing Trust in Autonomous Driving. In *IEEE International Smart Cities Conference*, pages 1–7.
- Ambreen, T., Ikram, N., Usman, M., and Niazi, M. (2018). Empirical research in requirements engineering: trends and opportunities. *Requirements Engineering*, 23:63–95.
- Asimov, I., Warrick, P. S., and Greenberg, M. H. (1984). *Machines that Think: The Best Science Fiction Stories about Robots and Computers*. H. Holt & Co.
- Bosch, J., Holmström Olsson, H., and Crnkovic, I. (2018). It takes three to tango: Requirement, outcome/data, and AI driven development. In Hyrynsalmi, S., Suominen, A., Jud, C., Wang, X., Bosch, J., and Münch, J., editors, *International Workshop on Software-intensive Business: Start-ups, Ecosystems and Platforms*, volume 2305, pages 177–192.
- Bosch, J., Holmström Olsson, H., and Crnkovic, I. (2021). *Engineering AI Systems: A Research Agenda*, pages 1–9. IGI Global.
- Canal, G., Borgo, R., Coles, A., Drake, A., Huynh, D., Keller, P., Krivić, S., Luff, P., ain Mahesar, Q., Moreau, L., Parsons, S., Patel, M., and Sklar, E. I. (2020). Building Trust in Human-Machine Partnerships. *Computer Law & Security Review*, 39:105489.
- Chakraborti, T., Sreedharan, S., and Kambhampati, S. (2020). The Emerging Landscape of Explainable AI Planning and Decision Making.
- Chien, S. and Morris, R. (2014). Space Applications of Artificial Intelligence. *AI Magazine*, 35(4):3–6.
- Gartner, Inc. (2020). Top Strategic Technology Trends for 2021. [www.gartner.com/en/newsroom/press-releases/2020-10-19-gartner-identifies-the-top-strategic-technology-trends-for-2021](http://www.gartner.com/en/newsroom/press-releases/2020-10-19-gartner-identifies-the-top-strategic-technology-trends-for-2021). Accessed: January 25, 2023.
- Georgievski, I. (2022). Towards Engineering AI Planning Functionalities as Services. In *Service-Oriented Computing – ICSOC 2022 Workshops*, Lecture Notes in Computer Science. Springer International Publishing.
- Georgievski, I. (2023). Conceptualising Software Development Lifecycle for Engineering AI Planning Systems. In *IEEE/ACM International Conference on AI Engineering – Software Engineering for AI*. Extended Abstract.

- Georgievski, I. and Aiello, M. (2015). HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222(0):124–156.
- Georgievski, I. and Aiello, M. (2016). Automated planning for ubiquitous computing. *ACM Computing Surveys*, 49(4):63:1–63:46.
- Georgievski, I. and Breitenbücher, U. (2021). A Vision for Composing, Integrating, and Deploying AI Planning Functionalities. In *IEEE International Conference on Service-Oriented System Engineering*, pages 166–171.
- Ghallab, M., Nau, D. S., and Traverso, P. (2004). *Automated planning: Theory & practice*. Morgan Kaufmann Publishers Inc.
- Grady, J. O., editor (2014). *System Requirements Analysis*. Elsevier, second edition.
- Green, A., Reji, B. J., Chris, C. M., Scala, E., Meneguzzi, F., Rico, F. M., Stairs, H., Dolejsi, J., Magnaguagno, M., and Mounty, J. (2019). Planning.Wiki - The AI Planning & PDDL Wiki. Accessed: January 24, 2023.
- Herschel, M., Diestelkämper, R., and Ben Lahmar, H. (2017). A Survey on Provenance: What for? What Form? What From? *The VLDB Journal*, 26(6):881–06.
- Höller, D., Behnke, G., Bercher, P., Biundo, S., Fiorino, H., Pellier, D., and Alford, R. (2020). HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. In *AAAI Conference on Artificial Intelligence*, pages 9883–9891.
- Karpas, E. and Magazzeni, D. (2020). Automated Planning for Robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):417–439.
- Leymann, F. and Roller, D. (1997). Workflow-Based Applications. *IBM Syst. J.*, 36(1):102–123.
- Ma, T., Ferber, P., Huo, S., Chen, J., and Katz, M. (2020). Online planner selection with graph neural networks and adaptive scheduling. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 5077–5084.
- McCluskey, T. L., Vaquero, T. S., and Vallati, M. (2017). Engineering Knowledge for Automated Planning: Towards a Notion of Quality. In *Knowledge Capture Conference*. Association for Computing Machinery.
- Muise, C., Pommerening, F., Seipp, J., and Katz, M. (2022). PLANUTILS: Bringing Planning to the Masses. In *International Conference on Automated Planning and Scheduling: System Demonstrations*.
- Munassar, N. M. A. and Govardhan, A. (2010). A comparison between five models of software engineering. *International Journal of Computer Science Issues*, 7(5):94–101.
- Myers, K. L. (1999). CPEF: A Continuous Planning and Execution Framework. *AI Magazine*, 20:63–69.
- Nau, D. S., Au, T. C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., and Yaman, F. (2003). SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, 20:379–404.
- Nilsson, N. J. (1984). Shakey the Robot. Technical Note 323, SRI International’s Artificial Intelligence Center.
- Olszewska, J. I. (2019). D7-R4: Software Development Life-Cycle for Intelligent Vision Systems. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 435–441. INSTICC, SciTePress.
- Pellier, D. and Fiorino, H. (2018). PDDL4J: A Planning Domain Description Library for Java. *Journal of Experimental and Theoretical Artificial Intelligence*, 30(1):143–176.
- Polyak, S. T. and Tate, A. (1998). Rationale in Planning: Causality, Dependencies, and Decisions. *The Knowledge Engineering Review*, 13(3):247–262.
- Silva, J. R., Silva, J. M., and Vaquero, T. S. (2020). Formal Knowledge Engineering for Planning: Pre and Post-Design Analysis. In Vallati, M. and Kitchin, D., editors, *Knowledge Engineering Tools and Techniques for AI Planning*, pages 47–65. Springer International Publishing.
- Studer, R., Benjamins, V. R., and Fensel, D. (1998). Knowledge Engineering: Principles and Methods. *Data & Knowledge Engineering*, 25(1–2):161–197.
- van Lamsweerde, A. (2009). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley Publishing, 1st edition.
- Vaquero, T. S., Silva, J. R., and Beck, J. C. (2011a). A Conceptual Framework for Post-Design Analysis in AI Planning Applications. In *Workshop on Knowledge Engineering for Planning and Scheduling*, pages 109–116.
- Vaquero, T. S., Silva, J. R., and Beck, J. C. (2011b). Acquisition and Re-Use of Plan Evaluation Rationales on Post-Design. In *Workshop on Knowledge Engineering for Planning and Scheduling*, pages 15–22.
- Vaquero, T. S., Silva, J. R., Tonidandel, F., and Christopher Beck, J. (2013). itSIMPLE: Towards an Integrated Design System for Real Planning Applications. *The Knowledge Engineering Review*, 28(2):215–230.
- Vulgarakis Feljan, A., Mohalik, S. K., Jayaraman, M. B., and Badrinath, R. (2015). SOA-PE: A service-oriented architecture for Planning and Execution in cyber-physical systems. In *International Conference on Smart Sensors and Systems*, pages 1–6.
- Weder, B., Barzen, J., Leymann, F., and Vietz, D. (2022). Quantum software development lifecycle. In *Quantum Software Engineering*, pages 61–83. Springer.
- Wu, Y., Chen, M.-H., and Offutt, J. (2003). UML-Based Integration Testing for Component-Based Software. In *International Conference on COTS-Based Software Systems*, pages 251–260.
- Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., et al. (2018). Accelerating the machine learning lifecycle with MLflow. *IEEE Data Eng. Bull.*, 41(4):39–45.