Software Engineering

Lecture 3

Plan-driven Software Development

Dr. Obuhuma James

Description

This topic gives a detailed coverage of the plan-driven methodology as applied to software development. Examples of software process models that embrace the concept of plan-driven development are also covered. Finally, the benefits and downsides of the plan-driven methodology are also covered.

Learning Outcomes

By the end of this topic, you will be able to:

- Broadly describe the plan-driven methodology as applied in Software Engineering.
- Describe examples of software process models that embrace the plan-driven methodology.
- Outline benefits and downsides of software process models that use plan-driven methodology.

Overview of Software Development Methodologies

The previous topic established a foundation for software development methodologies. Two methodologies were brought forward, namely, plan-driven and agile methodologies. Most of the existing software process models align to either of the two methodologies. It was however mentioned that some software process models embrace elements of both plan-driven and agile methodology. Such models are referred to as hybrid models.

The plan-driven methodology is a traditional approach to software development that follows a set of predefined steps while the agile methodology interleaves phases. Deciding on the appropriate methodology to use for a given software development project depends on a number of factors, for instance, the plan-driven methodology is appropriate if:

- Very detailed specification, analysis and design is required before moving to implementation.
- The system is large and requires larger development teams.
- The system has a long lifetime requiring more design documentation to communicate the original intentions of the system developers to the support team.

On the other hand, the agile methodology is appropriate if:

- Incremental delivery strategy is used, where you deliver the software to customers and get rapid feedback from them before starting the next increment.
- The system can be developed with a small co-located team who can communicate informally.

Other points to note while determining the methodology to use include:

- Agile methods rely on good tools to keep track of an evolving design
- If the development team is distributed or if part of the development is being outsourced, then you may need to develop design documents to communicate across the development teams.
- It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code
- If a system has to be approved by an external regulator, then you will probably be required to produce detailed documentation as part of the system safety case.

This topic focusses on exploring the plan-driven methodology by detailing its features, pros and cons, and the software process models that embrace the plan-driven approach.

Features of Plan-driven Methodology

Figure 1 shows the process of plan-driven development where each process activity must be fully completed before progression to the next phase. Output for each phase must be very comprehensive and clear since it acts as a basis for the next phase. For instance, the output from the requirements specification phase serves as input for the design and implementation phase.

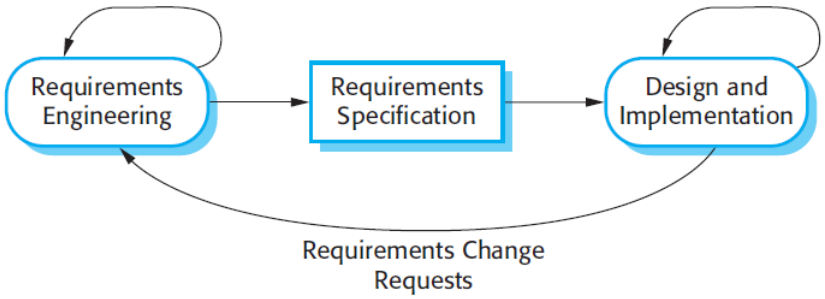

Figure 1. Plan-driven Methodology [1]

During the requirements engineering phase, an agreed requirements document that specifies a system satisfying stakeholder requirements is generated. At this point, requirements are usually presented at two levels of detail, namely, end-user needs, and system developer needs. Generally, end-users and customers need a high-level statement of the requirements while system developers need a more detailed system specification. The phase encompasses four development activities, namely, feasibility study, requirement elicitation and engineering, requirement specification, requirement validation.

Software implementation is the process of converting system specification into an executable system. It always involves software design and programming processes. Thus, software design is a description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, sometimes, the algorithms used. Designers do not arrive at a finished design immediately, but they develop the design iteratively. They add formality and detail as they develop their design with constant backtracking to correct earlier designs. Software coding follows the designs, by transforming them into an executable system.

Due to the conditions of the plan-driven approach, each phase (or activity) must be exhausted fully before proceeding to the next phase. Hence, a requirements specification document has to be completed to give room for the start of the design and implementation process. It is however worth noting that incremental development and delivery is also supported by the plan-driven methodology. This is based on the fact that requirements can be feasibly allocated, and design and development phase planned as a series of increments.

Plan-driven Process Models

A number of process models can be categorized under the plan-driven methodology depending on the nature of their system development process. This include the Waterfall model, the V-model, the Spiral model among others. The following subsections will try to explore three of these models as a way of helping the learner to comprehend the plan-driven nature of the models.

Waterfall Model

The waterfall model is a sequential approach, commonly expressed in a top – down linear fashion of phases. Each fundamental activity of a process is usually represented as a separate phase. As a plan-driven process model, the waterfall model requires planning and scheduling of all software development activities before commencement of development.

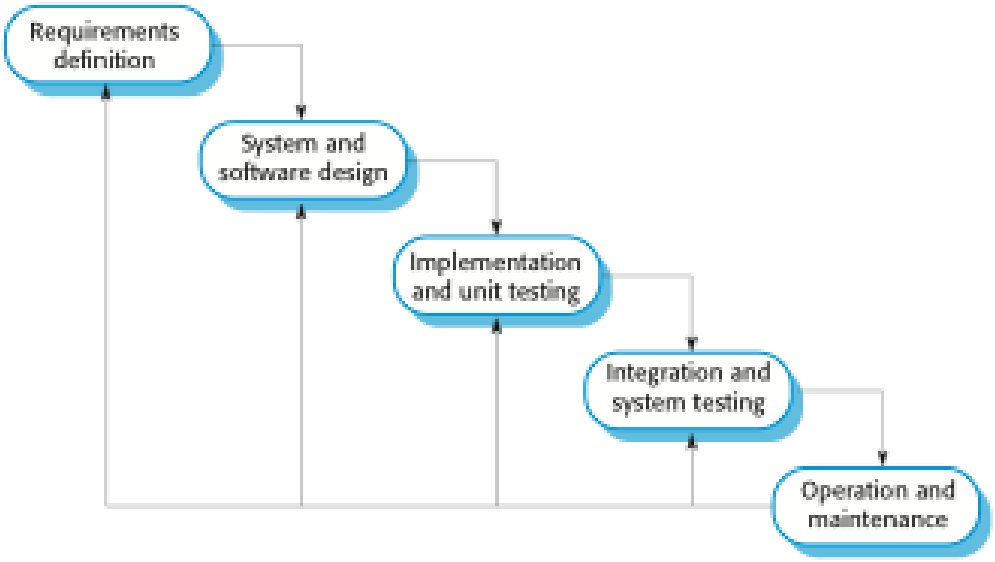

Figure 2. Waterfall Process Model [1]

Figure 2 shows the structure of the waterfall process model where the output of each phase acts as a clear input for the next phase. The phases of the waterfall model are as follow:

1. Requirements definition – the phase involves a meeting of the development team with project stakeholders aimed at identification of user needs. This is normally carried out as a feasibility study resulting into clear functional and non-functional requirements and their constraints. A system requirements document is generated as the end product of the phase.
2. System and software design – the phase involves the conversion of requirements into representation of architectural models. This is done by the development team. As part of the activities, prototypes are created as a form of communication of the desired system outcome. Accuracy in the designs is paramount since the phase is not revisited once the next phase starts, yet the design outcome is heavily depended upon throughout the entire process.

3. Implementation and unit testing – the phase sees the designs transformed into the actual software through coding. The various units of the software are tested as they are developed.
4. Integration and system testing – the entire system is tested with errors and bugs fixed. This could be achieved using different tools available at the disposal of the testers and developers to detect and fix defects in the software.
5. Operation and maintenance – the phase entail actual use of the software that leads to discovery of bugs and errors, arising of new user needs, and necessity for evolution. Thus, during the phase, the software gets updated (patched), to meet the ever-changing needs and environments. Furthermore, any arising bugs get fixed too.

This waterfall model is structured in such a manner that does not permit returning back to previous phases. Thus, mistakes made in any of the phases lead to increased costs of development, the project's timelines and developments teams' effort. Furthermore, this makes the process more rigid, hence, unsuitable for dynamic environment that require an iterative approach. The model, however, works fine for big government, military and other projects that utilise previous experience. Estimation of time, costs and complexity is foreseen in such projects.

Spiral Model

The Spiral model is yet another plan-driven process model that borrows its flow from the notion of the waterfall model. It however incorporates a risk analysis and an iteration element in each development phase as shown in Figure 3. The model ideally changed the phases of the waterfall model into many pieces of iteration processes aimed at reducing project risks. Thus, each iteration ensures that objectives, alternatives and constraints are determined, alternatives are evaluated, risks are identified and resolved, the product is developed, the next-level product is verified, and finally, the next iteration is planned.

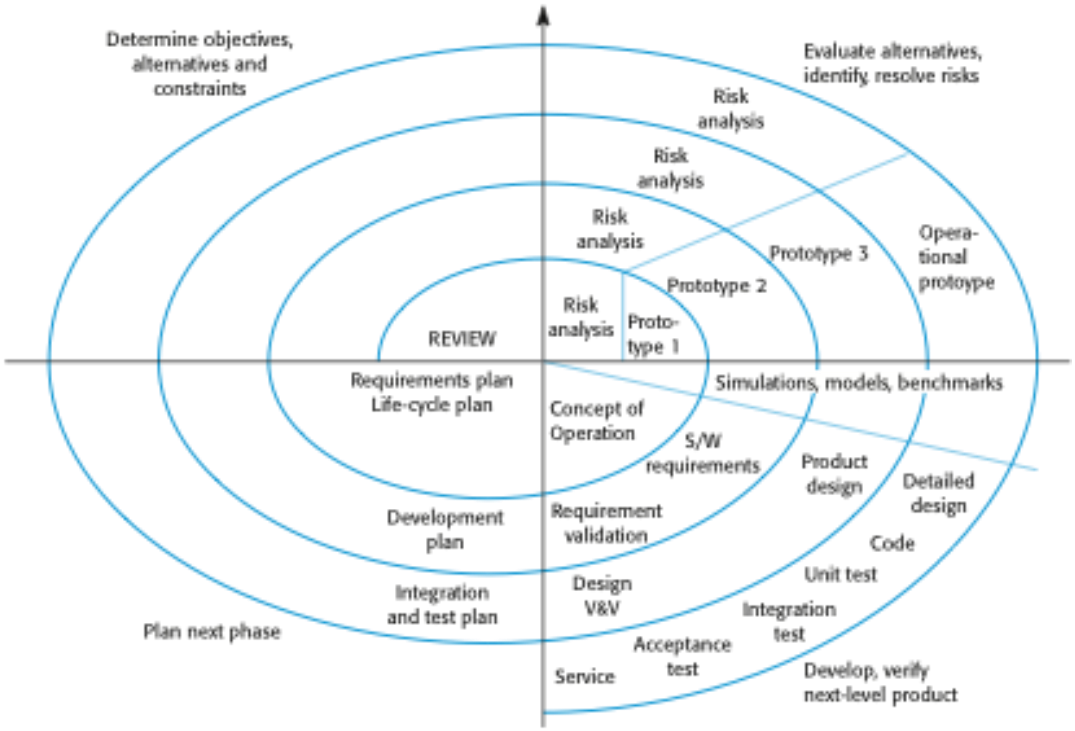

Figure 3. Spiral Process Model [1]

V-Model

The V-model is another plan-driven model represented as a v-shaped flow chart where it derives its name from. This is as shown in Figure 4. It comes as a refinement of the waterfall model, but with a focus on testing. Hence, it follows a sequence of activities like those in the waterfall model. As a plan-driven process model, each subsequent phase is begun upon the completion of the preceding phase.

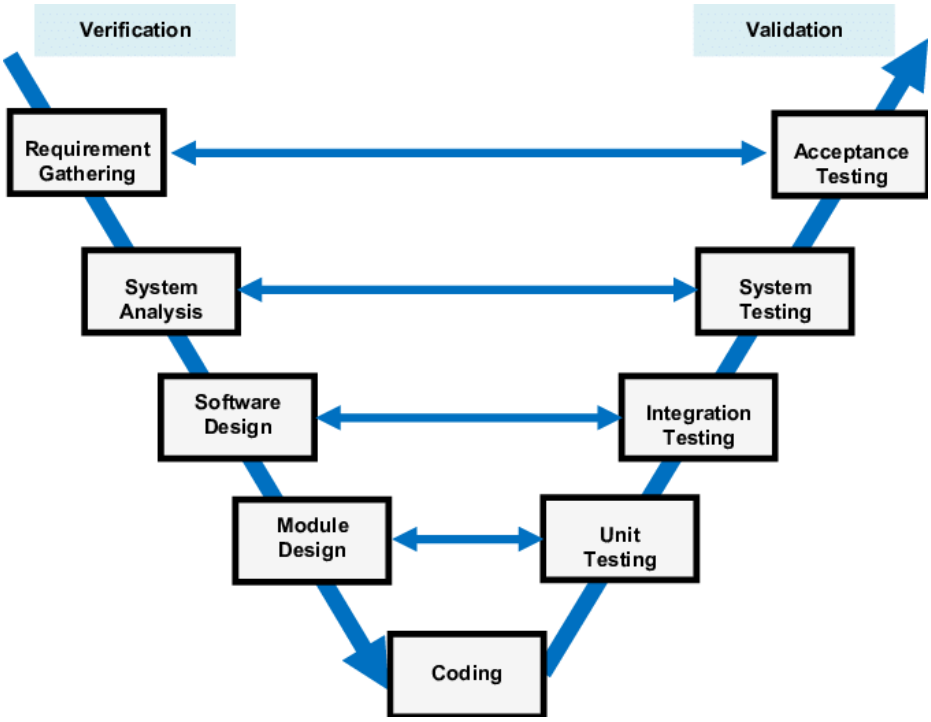

Figure 4. V- Model

The three models are examples of plan-driven models that could be used as a basis for categorizing any other process model. It is however worth mentioning that the current world of software development has really transformed the way things are done. A majority of current software process models use the plan-driven approach to define the set of steps to be followed, but they also include and agile approach to provide flexibility in the development by accepting iterative development. This will be explained further towards the end of the next topic.

Benefits and Downsides of the Plan-driven Methodology

Plan-driven methodologies are applicable where all the activities are planned first, and the progress is measured against the plan. This means that the methodology precisely depends on clear procedures. The Plan-driven development is more of a traditional development method that relies on the existence of a plan and architecture with a prime goal of production of quality software with better predictability of process.

Plan-driven methodologies cannot work in cases where requirements change over time, as businesses and the environment in which they operate change. Thus, plan-driven methodologies are not suitable for rapid development. In essence, plan-driven

methodologies advocate for change prediction using a framework. Thus, such an approach leads to resource wastage under rapid development environments. Finally, plan-driven methodologies limit on the development freedom due to the rigid set of activities to be followed during software development.

Summary

The topic covered the plan-driven methodology for software development. This was achieved by exploring the concept of plan-driven development that is rigid based on a predefined set of steps. The waterfall model, the V-model and the spiral model have been covered as examples of process models that embrace the plan-driven approach of software development. Finally, the pros and cons of the plan-driven approach have been outlined as a way of determining the suitability of the approach in software development. The next topic will cover the agile methodology as a way to try and provide solutions for the shortcomings of plan-driven approach.

Check Points

1. Discuss the salient features of the plan-driven methodology.
2. State and explain what makes the waterfall model be categorized as a plan-driven process model.
3. State and explain what makes the V-Model be categorized as a plan-driven process model.
4. State and explain what makes the spiral model be categorized as a plan-driven process model.
5. Discuss any two benefits of the plan-driven methodology.
6. Discuss any two downsides of the plan-driven methodology.

Learning Resources

Core Textbooks

1. Sommerville, I., Software Engineering, 9th Edition, Addison Wesley, 2016.

Other Resources

2. Gotterbarn, D., Miller, K., & Rogerson, S. (1997). Software engineering code of ethics. Communications of the ACM, 40(11), 110-118.

References

[1] Sommerville, I., Software Engineering, 10th Edition, Addison Wesley, 2016.

[2] Gotterbarn, D., Miller, K., & Rogerson, S. (1997). Software engineering code of ethics. Communications of the ACM, 40(11), 110-118.

[3] Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave macmillan.

[4] Kendall, K. E. and Kendall, J. E., Systems Analysis and Design, 8th Edition, Pearson, 2011.