The added text relates PEGS with the RE Reference Model

BLOG@CACM

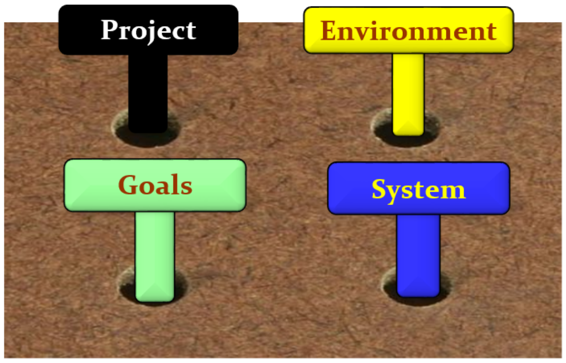
A Standard Plan for Modern Requirements

The PEGS approach: time to use a plan for specifying 21st century IT systems

By Bertrand Meyer
July 20, 2021

Requirements documents for software projects in industry, agile or not, typically follow a plan defined in a 1998 IEEE standard (IEEE 830-1998 [1]), "reaffirmed" in 2009. IEEE 830 has the merit of simplicity, as it fits in 37 pages of which just a few (competently) describe basic requirements concepts and less than 10 are devoted to explaining the standard recommended plan, which itself consists of 3 sections with subsections. Simplicity is good but the elementary nature of the IEEE-830 plan is just not up to the challenges of modern information technology. It is time to retire this venerable precursor and move to a structure that works for the kind of ambitious, multi-faceted IT systems we build today.

For the past few years I have worked on defining a systematic approach to requirements, culminating in a book to be published in the Fall, *Handbook of Requirements and Business Analysis*. One of the results of this effort is a standard plan, based on the "PEGS" view of requirements where the four parts cover Project, Environment, Goals and System. The details are in the book (for some of the basic concepts see a paper at TOOLS 2019, [2]). Here I will introduce some of the key principles, since they are already be used -- as various people have done since I first started presenting the ideas in courses and seminars (particularly an ACM Webinar, organized by Will Tracz last March, whose recording is [available on YouTube](), and another hosted by Grady Booch for IBM).



The starting point, which gives its name to the approach, is that requirements should cover the four aspects mentioned, the four "PEGS", defined as follows:

ZJVF World

A **Goal** is a result desired by an organization. R

A **System** is a set of related artifacts, devised to help meet certain goals. S SYS

A **Project** is the set of human processes involved in the planning, construction, revision and operation of a system.

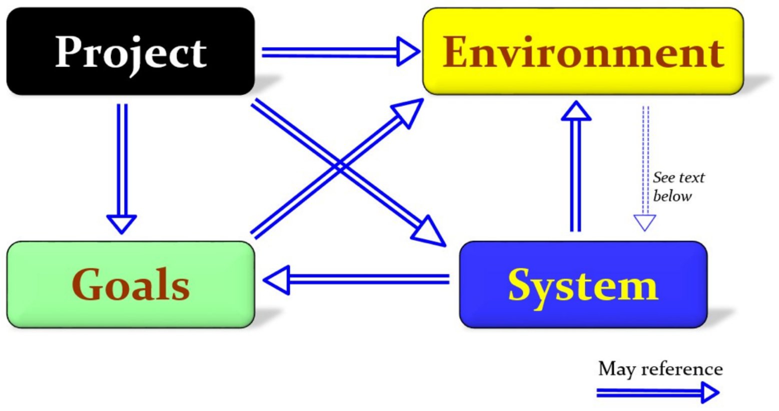
An **Environment** is the set of entities (such as people, organizations, devices and other material objects, regulations and other systems) external to the project and system but with the potential to affect the goals, project or system or to be affected by them. D ENV

The recommended standard plan consequently consists of four parts or *books*.

This proposed standard does not prescribe any particular approach to project management, software development, project lifecycle or requirements expression, and is applicable in particular to both traditional ("waterfall") and agile projects. It treats requirements as a project activity, not necessarily a lifecycle step. One of the principles developed in the book is that requirements should be treated as a dynamic asset of the project, written in a provisional form (more or less detailed depending on the project methodology) at the beginning of the project, and then regularly extended and updated.

Similarly, the requirements can be written using any appropriate notation and method, from the most informal to the most mathematical. In a recently published *ACM Computing Surveys* paper [3], my colleagues and I reviewed the various levels of formalism available in today's requirements approaches. The standard plan is agnostic with respect to this matter.

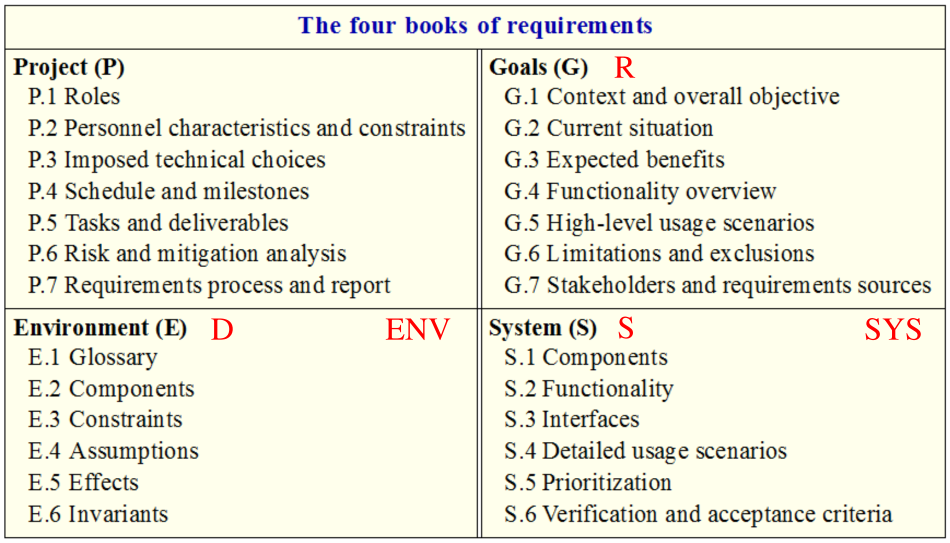
The books may reference each other but not arbitrarily. The permitted relations are as follows:



Note in particular that the description of the Goals should leave out technical details and hence may not refer to Project and System elements, although it may need to explain the properties of the Environment that influence or constrain the business goals. The Environment exists independently of the IT effort, and hence the Environment book should not reference any of the others, with the possible exception (dotted arrow in the figure) of effects of the System if it is to change the environment. (For example, a payroll IT system may affect the payroll process; a heating system may affect the ambient temperature.)

The multi-book structure of the recommended PEGS standard plan already goes beyond the traditional view of a single, linear "requirements document". The books themselves are not necessarily written as linear texts; with today's technology, requirements parts can and generally should be stored in a *requirements repository* which includes all requirements-relevant elements. A linear form remains necessary; it can be either written as such or produced by tools from elements in the repository.

The standard plan defines the structure of the four PEGS books down to one more level, *chapters*. For any further levels (*sections*), each organization can define its own rules. Books can also include *front and back matter*, including for example tables of contents, legal disclaimers, revision history etc., not covered by the standard structure. Here is that structure:



The four books of requirements	
Project (P)	**Goals (G)** R
P.1 Roles	G.1 Context and overall objective
P.2 Personnel characteristics and constraints	G.2 Current situation
P.3 Imposed technical choices	G.3 Expected benefits
P.4 Schedule and milestones	G.4 Functionality overview
P.5 Tasks and deliverables	G.5 High-level usage scenarios
P.6 Risk and mitigation analysis	G.6 Limitations and exclusions
P.7 Requirements process and report	G.7 Stakeholders and requirements sources
Environment (E) D ENV	**System (S)** S SYS
E.1 Glossary	S.1 Components
E.2 Components	S.2 Functionality
E.3 Constraints	S.3 Interfaces
E.4 Assumptions	S.4 Detailed usage scenarios
E.5 Effects	S.5 Prioritization
E.6 Invariants	S.6 Verification and acceptance criteria

It is meant to be self-explanatory, but here are a few comments on each of the books:

> One of the products of the requirements effort should be to help plan and manage the rest of the **Project**. This is the goal of the Project book; note in particular P.4 and P.5 covering tasks and deadlines. P.7 starts out at the beginning of the project as a blueprint for the requirements effort, and as this effort proceeds (stakeholder interviews, stakeholder workshops, competitive analysis, requirements writing ...) can be regularly updated to report on how it went. (This feature is an example of treating the requirements repository and documents as a living, dynamic asset, as noted above.)
>
> In the **Environment** book, *constraints* (E.3) are properties of the environment (the problem domain) imposed on the project and system. *Effects* (E.5) go the other way around, describing how the system may affect the environment. *Invariants* (E.6) do both. *Assumptions* (E.4) are properties that are taken for granted to simplify the construction of the system (for example, a maximum number of simultaneous users), as distinct from actual constraints.
>
> The **Goals** book is intended for a less technical audience than the other books: it must be understandable to decision makers and non-IT-savvy stakeholders. It includes a short summary (G.4) of functionality, a kind of capsule version of the System book trimmed down to the essentials. Note the importance of specifying (in G.6) what aspects the system is *not* intended to address. The Goals book can include some (G.5) usage scenarios expressed in terms meaningful to the book's constituencies and hence remaining at a high level of generality.
>
> *Detailed* usage scenarios will appear in the **System** book (S.4). It is important to prioritize the functions (S.5), allowing a reasoned approach (rather than decisions made in a panic) if the project hits roadblocks and must sacrifice some of the functionality.

A naïve but still widely encountered view of requirements is that they serve to "*describe what the system will do*" (independently of how it will do it). In the structure above, it corresponds to just one-fourth of the requirements effort, the System part. Work on requirements engineering in the past few decades has emphasized, among other concepts, the need to separate system and environment properties (Michael Jackson, Pamela Zave) and the importance of goals (Axel van Lamsweerde).

The plan reflects this richness of the requirements concept and I hope it can help many projects produce better requirements for better software.

References

[1] IEEE 830-1998, available here.

[2] Bertrand Meyer, Jean-Michel Bruel, Sophie Ebersold, Florian Galinier and Alexandr Naumchev: *The Anatomy of Software Requirements*, in *TOOLS 2019*, Springer Lecture Notes in Computer Science 11771, 2019, pages 10-40.

[3] Jean-Michel Bruel, Sophie Ebersold, Florian Galinier, Manuel Mazzara, Alexander Naumchev and Bertrand Meyer: *The Role of Formalism in System Requirements*, in *Computing Surveys* (ACM), vol. 54, no. 5, June 2021, pages 1-36, DOI: https://doi.org/10.1145/3448975 , preprint available here.

Bertrand Meyer *is professor and provost at the Schaffhausen Institute of Technology (Switzerland) and chief technology officer of Eiffel Software (Goleta, CA).*

No entries found