Hestia

A Software Product Line Architecture for reducing maintenance effort

Marcelo Schmitt Laser

Outline

1. Motivation - Is maintenance effort an issue?
2. Goals - How to reduce maintenance effort?
3. Software Product Lines - How do SPLs help?
4. Architecture - Why is Hestia important?
5. Roadmap - Where is the project going?

Conventions used in this presentation

- Effort: Time expended, monetary cost, number of human resources required, amount of computational resources required.
- Project: Any non-trivial software-intensive projects.
- Deployment: All the actions required to make a finished, independent software component work within the scope of a project.

Motivation

Is maintenance effort an issue?

Maintenance

What is maintenance comprised of?

- **Creating and updating project documentation.**
- Implementing and deploying new software components.
- Updating and refactoring existing software components.
- Briefing and training project members.

Maintenance

Proper project documentation is paramount to the reduction of maintenance effort.

- Insufficient or deprecated project documentation can increase the effort required for briefing and training project members by orders of magnitude.
- Insufficient project documentation is a known cause of project failures.
- Excessive project documentation can be a barrier to the adoption of good practices and can be difficult to comprehend.

Maintenance

What is maintenance comprised of?

- Creating and updating project documentation.
- **Implementing and deploying new software components.**
- Updating and refactoring existing software components.
- Briefing and training project members.

Maintenance

The implementation and deployment of new software components is a constant cause of problems.

- Project requirements change, often rapidly, over time. Keeping up with these changes usually requires the creation of new software components.
- Implementing new software components for a mature project can be made simpler if the project has some kind of template or framework to be followed.
- Deploying new software components can often cause the “cascading” of bugs through existing software components.

Maintenance

What is maintenance comprised of?

- Creating and updating project documentation.
- Implementing and deploying new software components.
- **Updating and refactoring existing software components.**
- Briefing and training project members.

Maintenance

Updating and refactoring existing software components is necessary for any project.

- Even a mature, established software will require updates to keep up with technological advances.
- When bugs appear due to the deployment of new software components, often the best solution is the refactoring of parts of the old components.
- As project requirements change, the best solution is often to update old components to execute slightly different functions.

Maintenance

What is maintenance comprised of?

- Creating and updating project documentation.
- Implementing and deploying new software components.
- Updating and refactoring existing software components.
- **Briefing and training project members.**

Maintenance

Even if a project's team does not change over a long period of time, it is common for project members to perform different functions over the course of a project.

- Changes in the project's team (members dropping out, new members coming in) require training so that all members are up-to-date with the project's status.
- When project members switch functions, it is important for the member previously performing that function to brief them in on the required tasks.
- Depending on the type of project, training can become one of the greatest challenges in managing that project.

Maintenance

Maintenance can often become the most costly part of a project if it is not managed properly.

Goals

How to reduce maintenance effort?

Goals

Although there are several methods for reducing maintenance effort, such as proper business practices and healthy workplace environment, this project focuses on those that have to do with Software Architecture.

- **Proving the importance of project documentation.**
- Identifying the right balance in the amount of project documentation.
- Keeping track of the responsibilities of team members.
- Creating a project framework to facilitate implementation.
- The importance of a project’s implementation cannot be ignored.

Goals

It is often the case that project documentation is lacking due to the project members not believing that it is important.

- Proving the importance of project documentation is an excruciatingly difficult task, as industrial studies in the subject matter require enormous cooperation from industrial partners.
- Misconceptions and myths regarding project documentation can cause many software professionals to be predisposed to ignore documentation.
- Excessively theoretical formal education causes software professionals to become averse to this field of project management.

Goals

Although there are several methods for reducing maintenance effort, such as proper business practices and healthy workplace environment, this project focuses on those that have to do with Software Architecture.

- Proving the importance of project documentation.
- **Identifying the right balance in the amount of project documentation.**
- Keeping track of the responsibilities of team members.
- Creating a project framework to facilitate implementation.
- The importance of a project’s implementation cannot be ignored.

Goals

Despite the importance of project documentation, it is important to recognize limitations on time and financial support.

- Finding the lines between “insufficient documentation”, “proper documentation” and “excessive documentation” can be difficult.
- Different projects will require different levels of documentation.
- A minimalistic approach to architectural documentation can help both in its adoption and in its comprehension.

Goals

Although there are several methods for reducing maintenance effort, such as proper business practices and healthy workplace environment, this project focuses on those that have to do with Software Architecture.

- Proving the importance of project documentation.
- Identifying the right balance in the amount of project documentation.
- **Keeping track of the responsibilities of team members.**
- Creating a project framework to facilitate implementation.
- The importance of a project’s implementation cannot be ignored.

Goals

It is important to bear in mind that a project is not only comprised of its artifacts, but also of its members.

- Proper recognition of project members' work can serve as an incredible motivation for continued performance.
- Identification of responsibilities of each project member is necessary to quickly evaluate and tackle faults in the project.
- Keeping track of project members' tasks can greatly facilitate communication.

Goals

Although there are several methods for reducing maintenance effort, such as proper business practices and healthy workplace environment, this project focuses on those that have to do with Software Architecture.

- Proving the importance of project documentation.
- Identifying the right balance in the amount of project documentation.
- Keeping track of the responsibilities of team members.
- **Creating a project framework to facilitate implementation.**
- The importance of a project’s implementation cannot be ignored.

Goals

Having a solid architectural framework from the beginning of a project can greatly ease the effort of the implementation phase.

- Architectural documents are the primary sources for developers to understand a project.
- A poorly designed architecture increases the difficulty of project evolution, as it causes greater portions of code to require refactoring.
- The presence of templates for architecturally similar components can facilitate the implementation of new components through reuse techniques.

Goals

Although there are several methods for reducing maintenance effort, such as proper business practices and healthy workplace environment, this project focuses on those that have to do with Software Architecture.

- Proving the importance of project documentation.
- Identifying the right balance in the amount of project documentation.
- Keeping track of the responsibilities of team members.
- Creating a project framework to facilitate implementation.
- **The importance of a project's implementation cannot be ignored.**

Goals

It is often the case in Software Engineering research that there is a primary emphasis on modelling and process, and little to no emphasis on implementation.

- Ultimately, the maintenance of a software-intensive project is handled by the developers.
- Having a development team that is capable of quickly handling code refactoring can simplify the management of the entire project.
- Documentation that cannot be eventually translated to code (regardless of how many steps it is removed from it) is often useless to the project.

Goals

The primary goal of the Hestia research is to facilitate project maintenance by proposing minimalistic documentation guidelines.

Software Product Lines

How do SPLs help?

Software Product Lines

“A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.” - Software Engineering Institute

- **By employing SPL techniques, products can easily be tailored to the specific needs of different users without the need for code refactoring.**
- The component-based view that SPLs offer for software architecture is propitious to distributed development.
- The maintenance of a single product of an SPL necessarily causes the entire line to be kept up-to-date, often resuming the maintenance tasks of several different products to a single task.

Software Product Lines

SPLs are based on the centuries-old notion of Product Lines, or Product Families, where several different products can be created from a single pool of shared resources.

- It is indispensable to bear in mind that SPLs do have a break-even point, and projects that focus on single products may not benefit from SPL techniques.
- The tailoring of products to specific users can increase the value of the product, but will often increase the cost as well. SPL seeks to minimize that cost.

Software Product Lines

“A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.” - Software Engineering Institute

- By employing SPL techniques, products can easily be tailored to the specific needs of different users without the need for code refactoring.
- **The component-based view that SPLs offer for software architecture is propitious to distributed development.**
- The maintenance of a single product of an SPL necessarily causes the entire line to be kept up-to-date, often resuming the maintenance tasks of several different products to a single task.

Software Product Lines

The need for teams to work together for a majority of a project can limit its advance, as team members will be unable to work individually when waiting for the tasks of others to be finished.

- SPL and component-based techniques facilitate the encapsulation of tasks so that team members can work individually, as well as in groups.
- The development of independent components can often be done by professionals with little to no knowledge of the project, easing the transition for new project members.

Software Product Lines

“A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.” - Software Engineering Institute

- By employing SPL techniques, products can easily be tailored to the specific needs of different users without the need for code refactoring.
- The component-based view that SPLs offer for software architecture is propitious to distributed development.
- **The maintenance of a single product of an SPL necessarily causes the entire line to be kept up-to-date, often resuming the maintenance tasks of several different products to a single task.**

Software Product Lines

Due to SPL forming products out of a common pool, the maintenance of these products is done not on the products themselves, but on the components in this pool.

- Upon updating a single component, every product in the family that makes use of it will be updated as well.
- By keeping different versions of components and forking development of future versions, different components can be added to the component pool.
- Errors are less likely to appear when more effort is placed into a single component, rather than several implementations of the same functionality, and can more easily be resolved when they are compartmentalized to a single location.

Software Product Lines

SPL techniques have a large emphasis on maintenance and can be used to facilitate maintenance tasks.

Architecture

Why is Hestia Important?

Architecture

Software architecture documentation is the primary source for developers to implement a software system, and is one of the most important documents in a software project.

- **Despite the importance of architecture documentation to implementation, a majority of SPL research does not go into implementation in much depth.**
- The Hestia research seeks to propose an architectural framework for creating new SPL architectures that will be easy to implement and maintain.
- Through Hestia, it should be possible to create loosely-coupled component-based system architectures regardless of the domain.

Architecture

The vast majority of the SPL research found by the authors focuses on SPL modelling, with very little emphasis on implementation.

- There is very little up-to-date information on how to implement variability mechanisms, and what little there is is often too complicated for entry-level professionals.
- While there is a vast number of representational forms and tools for SPL, such as Feature Diagrams and FeatureIDE, there are few sources that explain the SPL architectural concepts in simple terms.

Architecture

Software architecture documentation is the primary source for developers to implement a software system, and is one of the most important documents in a software project.

- Despite the importance of architecture documentation to implementation, a majority of SPL research does not go into implementation in much depth.
- **The Hestia research seeks to propose an architectural framework for creating new SPL architectures that will be easy to implement and maintain.**
- Through Hestia, it should be possible to create loosely-coupled component-based system architectures regardless of the domain.

Architecture

This research project hopes to propose an architectural framework that will be self-contained and easily accessible to entry-level professionals, so that its adoption may be less costly.

- The Hestia research project began with a group of first-through-third semester Undergraduate students of computer science and related fields. Due to that, it has always been tailored for entry-level professionals with little to no experience in real projects.
- The end-goal of the Hestia project is to provide a framework for the creation of SPL architectures that will be usable by anyone, from training through documentation and development, all the way to deployment and system evolution.

Architecture

Software architecture documentation is the primary source for developers to implement a software system, and is one of the most important documents in a software project.

- Despite the importance of architecture documentation to implementation, a majority of SPL research does not go into implementation in much depth.
- The Hestia research seeks to propose an architectural framework for creating new SPL architectures that will be easy to implement and maintain.
- **Through Hestia, it should be possible to create loosely-coupled component-based system architectures regardless of the domain.**

Architecture

Keeping in mind with the question of how to reduce maintenance effort, the Hestia research project seeks to study primarily the mechanisms of documentation and development that may make the evolution of a system simpler.

- At the forefront of the Hestia research are the developers. Development re-work should never be necessary, and we believe that if the developers are capable of doing their job more easily, the project will be less prone to error.
- Some of the primary terms of research of the Hestia Project are component coupling and dependency, system granularity, implementation of variability mechanisms, and automatic product generation.

Architecture

Hestia seeks to make the developer’s life easier by use of proper architectural mechanisms, reducing the number of monotonous tasks and helping him do a better, simpler job.

Roadmap

Where is the project going?

Roadmap

Hestia is in its very early stages, and does not yet have a formal definition. It was created in order to simplify a single project on which the authors were working at the time, and grew to become an architectural project in its own right.

- **The first version of Hestia was successfully applied in the PLeTs project.**
- Work is being conducted on the creation of an IDE to aid in the research and application of Hestia.
- The validation of Hestia in industrial projects is pending, but viable opportunities have already arisen.
- “Toy” systems are being created for the formalization of the architecture in a comprehensible format, suitable for educational purposes.

Roadmap

The first version of Hestia was not yet formally called by this name, as it was a prototype of little ambition made for the architecture of the PLeTs system.

The publication referent to this primary work can be found in:

http://ksiresearchorg.ipage.com/seke/seke15paper/seke15paper_57.pdf

Roadmap

Hestia is in its very early stages, and does not yet have a formal definition. It was created in order to simplify a single project on which the authors were working at the time, and grew to become an architectural project in its own right.

- The first version of Hestia was successfully applied in the PLeTs project.
- **Work is being conducted on the creation of an IDE to aid in the research and application of Hestia.**
- The validation of Hestia in industrial projects is pending, but viable opportunities have already arisen.
- “Toy” systems are being created for the formalization of the architecture in a comprehensible format, suitable for educational purposes.

Roadmap

Research is being conducted on the creation of an IDE with the primary purpose of teaching and automating the tasks related to Hestia. Thus far we have proposed the Hephaestus tool for product derivation, and the Atlas tool for feature modeling, though both are in very initial stages and not yet usable. Both of these works have been published on the ELA-ES conferences of 2015 and 2016, but are unavailable for online. The authors will, however, be happy to provide the papers in full to anyone interested.

Roadmap

Hestia is in its very early stages, and does not yet have a formal definition. It was created in order to simplify a single project on which the authors were working at the time, and grew to become an architectural project in its own right.

- The first version of Hestia was successfully applied in the PLeTs project.
- Work is being conducted on the creation of an IDE to aid in the research and application of Hestia.
- **The validation of Hestia in industrial projects is pending, but viable opportunities have already arisen.**
- “Toy” systems are being created for the formalization of the architecture in a comprehensible format, suitable for educational purposes.

Roadmap

While the very first work conducted using Hestia was in a research setting for a tool applied in the industry, the authors understand that further validation of the architecture is necessary in larger industrial settings, so as to prove its versatility and scalability. Opportunities for such research have already been presented to the authors, and they are looking into how best to make use of these opportunities.

Roadmap

Hestia is in its very early stages, and does not yet have a formal definition. It was created in order to simplify a single project on which the authors were working at the time, and grew to become an architectural project in its own right.

- The first version of Hestia was successfully applied in the PLeTs project.
- Work is being conducted on the creation of an IDE to aid in the research and application of Hestia.
- The validation of Hestia in industrial projects is pending, but viable opportunities have already arisen.
- **“Toy” systems are being created for the formalization of the architecture in a comprehensible format, suitable for educational purposes.**

Roadmap

The authors are seeking individuals interested in helping conduct studies in the creation of “toy” systems to aid in the formalization of Hestia. While these systems will not serve as proper validation of the system (such validation must be done through industrial projects), the authors believe a more didactic approach to the definition will be beneficial for its understanding by the public.

Thank you very much for your attention

Contact: marcelo.laser@gmail.com