Eclipse Requirements Council: Themes and Priorities

December 15, 2004

Introduction

This document captures the inputs to the Eclipse Requirements Council as of this date. The inputs are from three major sources:

1. Strategic Members of the Eclipse Foundation,
2. the Project Management Committees (PMCs), and
3. the general membership through the input collected at Eclipse members meeting held in Fort Worth, Texas.

The themes were generated by collecting requirements from the above sources and then synthesizing themes from the various inputs.

Fundamental Principles

These are our fundamental principles, which span all of the themes outlined below.

- Eclipse projects provide exemplary (show by example) implementations and extensible frameworks. Examples target open source implementations such as CVS, Ant, JBoss, JOnAS, Mono, and Tomcat.
- The Eclipse community will relentlessly re-factor code to improve components and make them available for public consumption.
- We will aim for release-to-release binary compatibility for API clean plug-ins for at least the last major release. For new projects, this binary compatibility requirement may be relaxed for initial two releases as the APIs are firmed up.
- The platforms supported by each project are determined by the specific project. Each project must keep its support for all supported platforms current and publicly announce otherwise.

Themes and Priorities

These themes are not in priority order.

1. Scaling Up

This refers to the need for Eclipse to deal with development and deployment on a larger and more complex scale. Increasing complexities arise from large development teams distributed in different locations, large source code bases and fragile build environments that have been developed incrementally over time, the dynamic nature of new source code bases and their interaction with configuration management, and build environments involving many different tools and build rules. For example, static analysis in development environment should be incremental instead of computing the dependencies from scratch every time.

This requires:

- Performance improvements in memory footprint, user perceived response times, and start-up times as the complexity and number of projects, files, users, and plug-ins grow (10X-100X over the next two years). For example, minimal RCP and Update Manager
- Being able to deal with extremely large projects and large workspaces. This may include a more efficient way to simultaneously manage multiple workspaces.

- A performance profiling and instrumentation infrastructure to help plug-in developers assess, root-cause, and fix performance issues. TPTP can be leveraged for this.

2. Enterprise Ready

Development: We need changes to Eclipse to allow it to be better used by large development organizations. As the size of a development organization grows, the manner in which an organization as a whole uses Eclipse changes. For example,

- Which features/capabilities get installed and are visible.
- Provide an ability for installed features/capabilities to be locked by an enterprise for its users
- Allow users to share their preferences and make sure that all user configuration settings are stored in a preference file
- Make it easier to configure workspaces (e.g. in order to be distributed to team members)
- Improve development time integration across development roles
    - Provide support for development time work flow. (e.g. an extensible process flow that could enforce a series of activities for a code branch commit – code review, statistics, unit test).
- This would include support for more roles and tools (e.g. extend our current Java only development model to cover defects, test cases, requirements)
    - Make this integration more seamless
- Improve configurability (e.g. of capabilities) to allow for the definition of unique roles
- Improve integration between Eclipse's build system and external build systems and deployment tools
- Provide a mechanism that makes it easier for Eclipse tools that do not require a full "headless" Eclipse to execute. This includes the notion of a command line interface for Eclipse

Deployment: We need changes to Eclipse to enable deployment of Eclipse based applications across the enterprise. This includes monitoring capabilities, developing models of the components and interactions in the application delivery chain, enhancing support of logging, tracing, and statistical models to enable prompt troubleshooting in a distributed environment, increasing interoperability with enterprise security infrastructure, and report generation. Develop a robust mechanism to ensure that products from different companies that work well separately on an Eclipse package work well together.

Manageability: JMX is rapidly becoming a standard for Java developers who want to incorporate manageability into their applications during development time. The new J2SE 1.5 JDK includes JMX support. Support for JMX design patterns which enable developers to model manageability and drive transformation to JMX MBeans and other management technologies. Additionally, Eclipse support for JMX creation, via wizards, code assist or other tooling, and monitoring would help in automating the management instrumentation process.

3. Design for Extensibility: Be a Better Platform

Within the Eclipse community, many development projects are defining new development platforms on top of the Eclipse Platform project. Concrete examples include the Web Tools Platform Project and the Test and Performance Platform Tools Project. It is recognized, however, that some function is not strictly required by the base but is important to exist at the base platform level in order to enable other platforms to succeed. Examples include common undo stack functionality, an extensible navigator, improved command framework, etc.

Opening the internal JDT (UI) interfaces would enable tools to seamlessly facilitate and interact with the JDT core and UI layers. Examples are: the parsing and AST functionality for static code analysis within and outside of Eclipse, extensions of the syntax (SQLJ), having control over and re-using the Java editor (SQLJ, protected code blocks, Java editor as one page of a multi-page editor, ...).

Loosening the strong file orientation by providing an abstraction layer of logical objects would allow to extend the proven Eclipse features to tools working on a higher abstraction level. One example would be the Marker and Quick fix capabilities. In this connection a less restrictive structuring of projects would be desirable (some tools would like to structure and group projects in a more hierarchical way).

4. Embedded Development

Instead of plug-in providers each developing their own plumbing to launch, profile, debug, and manage remote agents running on target, the requirement is for Eclipse platform to provide a consistent, uniform plumbing to address this infrastructure need. These capabilities can be leveraged for:

- Target OS independent debugging.
- Target OS independent profiling.
- Loading and launching on remote targets

We need a generic control centre - API to configure, setup, connect and monitor any type of target, node or network.

5. Rich Client Platform

The Eclipse RCP is a Java-based application framework for the desktop. Building on the Eclipse runtime and the modular plug-in story, it is possible to build applications ranging from command line tools to feature-rich applications that take full advantage of SWT's native platform integration and the many other reusable components. This theme includes work:

- To provide PDE support for developing and deploying RCP-based applications in enterprise environments
- To improve our ability to construct rich user interfaces through improvements to SWT and JFace
- To evolve RCP for embedded devices
- To improve authentication and security (user authentication and credentials, role based security, and role-based plug-in loading)
- To improve the capabilities of the UpdateManager to manage deployed rich application desktops.

6. Simple to Use

The Eclipse components need to not only provide the features that advanced users demand, but also be something that most users find simple to use. This theme includes ease-of-use reviews of existing features, and work that helps make Eclipse-based products simple to use for users with widely-varying backgrounds and skill sets.

Focus on ease of use through enhanced user documentation, tutorials, white papers, demonstrations, and a wide range of enhancements to the user interface to streamline basic processes and clarify concepts and terminology. For example, BIRT has a goal of making a user productive in 15 minutes.

7. Enable Consistent Multi-language Support

The original vision of Eclipse was to accelerate the creation of IDEs. Experience with the CDT and COBOL projects have shown that there is still a lot of work to do to make it simpler to create language-specific IDEs. Our vision is to create an environment where a limited number of well-defined extension points for compilers, parsers, debuggers, building, launching, etc. will allow language IDE developers to simplify the process of creating Eclipse-based offerings.

Plug-in providers have a strong need for an abstract development toolkit where a variety of plug-ins (e.g., editor, managed build system, debugger) can be plugged in for all development environments including Java. This is more efficient than integrating each plug-in with each separate development environment. Focus on an abstract development toolkit improves componentization and consistency across development environments, makes each component within the development environment more robust, and adds flexibility to increase the reach of Eclipse to a variety of development environments (e.g., Fortran, COBOL). This opens up 1:1 relationship between add-in component and development environment into a richer m:n relationship.

This support includes the following features:

- Make it easier to create language specific tools in a consistent way. Consistent generic components / APIs to add languages.
- Generic APIs – not Java-specific
- Ability to create tools for other languages
- Prescribe model for language toolkit which should provide public APIs sufficient for tools builders.

8. Appealing to the Broader Community

This theme includes work that grows deeper roots into the various OS-specific communities, spreads Eclipse to additional operating environments, virtual machines, application development and deployment lifecycles, vertical market-specific frameworks and builds bridges to other open source communities. This can be accomplished through additional documentation and more usable documentation, engineering engagements, marketing and an extensive outreach program incorporating trade shows, user groups, and press and analyst relations opportunities.

- **Improve consistency among implementations on Windows and Linux**

  The plug-in providers provided examples of different behaviours of Eclipse on Windows and Linux. For example, in one version the buttons would wrap around in a GUI when in other the buttons would be chopped off. This increases code bloat and complexity of testing a variety of combinations. If the behaviour of Eclipse on a variety of operating systems was consistent, this would significantly ease the burden on plug-in providers. Over 80% of Eclipse downloads are onto Windows, followed by ~20% onto Linux, and a very small fraction are onto operating systems such as AIX, Solaris, and HP-UX.

- **Swing – SWT Interoperability**

  Allow existing Swing/AWT applications to work seamlessly inside the SWT (Eclipse) environment. While the ability to make Swing components function within SWT UI's was made available in Eclipse 3.0, the ability to integrate complete applications requires additional efforts.

  For true interoperability between Eclipse and other Swing-based development environments, Eclipse also needs to enable an SWT application to run within a Swing environment.

- **J2SE 5 support with JDT**

- **Provide basic web services tools**

  Eclipse should provide basic tools and frameworks for supporting the construction, deployment and management of web service applications. Example tools include: UDDI browser, XSL/T editor, and WSDL tools.

APPENDIX

This section contains valuable inputs which were received by the Requirements Council but which are out of scope for this document. Our intent with this document is to focus on common themes for development across the Eclipse projects. These comments, although very helpful, are primarily requirements of the Eclipse projects laid upon their own build infrastructure and also upon the IT infrastructure provided by the Eclipse Foundation.

Eclipse Development Process Improvements

Robust distributed build processes:

Improve build processes to enable multiple organizations to build and test for a variety of hardware platforms, operating systems, JVMs, and other virtual machines. Work out a better process for independent contributors to synchronize contributions with the PMCs and core development teams. Today the process for synching with project teams is somewhat ad hoc and does not account for contributions that come in outside of the core build cycles. It's also difficult for contributors who are working independently of the core development team to get their code contributions built and tested prior to public availability on the eclipse.org web site.

Release Engineering:

- Improve the automated test suites. This includes more complete coverage for the Platform as well as ensuring all other subprojects have a set of automated test cases. TPTP currently uses its own infrastructure to define the tests and publish results to a CVS repository on a regular weekly basis. Perhaps, this infrastructure can be leveraged across Eclipse.
- Ensure all subprojects are available from the Eclipse update manager site.
- Provide a utility to help assess whether or not a plug-in is “API clean”

Increase roadmap visibility:

Make Eclipse project roadmaps visible, up-to-date and longer. The Eclipse community needs clearer visibility on what features are coming over a 12 month time horizon. This will allow Eclipse developers to better plan contributions, Eclipse Add-in providers to better determine where to innovate, and Eclipse users to better determine whether their requirements will be satisfied by Eclipse. Today many available roadmaps are out-of-date and provide no more than 3 months of visibility.

Project Start-up Guidelines:

When new projects are started in the Eclipse community, guidelines and "cheat sheets" are needed to help those projects during the startup process with regards to required Eclipse Foundation infrastructure and Eclipse community infrastructure. The overall goal of these guidelines is to enable new projects to come online faster and easier, and to reduce the overhead on existing project members to help in this process. The guidelines should include specific actions and how these should be accomplished (for example, who to contact and what information is needed in order to get the required CVS repository up and running). Topics would include:

- Process for establishing committers and completing all required legal steps
- Code management recommendations and steps
- Creating a CVS project repository for the project
- Project web site pages recommendations, structure conventions, and update process
- Build process recommendations and steps
- Community infrastructure recommendations and steps (e.g. newsgroups)

Contributors

Name	**Member**
Paul Clenahan	Actuate
John Kellerman	IBM Corporation
Anurag Gupta	Intel
Derrick Keefe	QNX Software
Par Emanuelsson	Ericsson AB
Philip Ma	Hewlett-Packard Company
Michael Bechauf	SAP
Rich Main and Todd Williams	Add-In Provider Board Representatives
John Wiegand	Platform PMC
David Williams	Web Tools PMC
Mike Norman	Test&Performance PMC

Acknowledgements

Anurag Gupta (Intel) wrote the first draft of the themes.

Mike Milinkovich (EF) and John Kellerman (IBM) reviewed the draft and provided feedback – their input has been incorporated into a second draft.

The second draft was reviewed by Paul Clenahan (Actuate), John Kellerman (IBM), Anurag Gupta (Intel), Derrick Keefe (QNX), Par Emanuelsson (Ericsson AB), and Philip Ma (HP) and incorporated into a third draft.

The third draft was reviewed at the Eclipse Requirements Council F2F meeting on Nov 30 and that feedback has been incorporated by Anurag Gupta into the fourth draft .