Building Durable Enterprise Architectures

Extending Build versus Buying Decision Frameworks with Open Source Options

Agenda

9:00am Introduction
9:15am History of the Open Source Concept
9:45am Essentials of OSS Licensing
10:00am Basic economics of OSS
10:45am Coffee
11:00am Strategic Assets, Commodities, Liabilities
12:15pm The OSS Value Proposition in 2005
12:30pm Lunch
1:30pm Open Standards
2:00pm Discussion : OSS in Government Departments
2:30pm A Realistic Approach to Enterprise Architecture
3:00pm Coffee
3:15pm Using OSS Commodities
3:45pm Building OSS Assets
4:00pm Developing a Practical Roadmap for OSS Adoption
4:15pm Open Discussion & Wrap Up

Introduction

- Names and roles
- Current level of understanding of the Open Source concept
- Your expectations for today's workshop
  - To be able to assess whether OSS can add value to your organization?
  - How to go about evaluating OSS software?
  - An overview of those OSS components that are relevant in practice?
  - To understand the risks of going down a path of OSS adoption?
  - To understand the support process for OSS software?
  - To be able to explain the OSS value proposition to your manager?
  - To understand why some people in your team keep talking about OSS?
  - Other?
- Your goals for the next 12 months regarding OSS

Acknowledgements

Much of the material in this presentation has its origin in material produced by key figures in the Open Source community. Hence the slides contain extensive references.

On many occasions I found that Bruce Perens *the guy who announced "Open Source" to the world, who published his first Open Source program in 1987, who has been at the center of this revolution from day one* already had the perfect words to explain the essence of Open Source and its economic impact. Wherever in this presentation you find paragraphs in italics without an explicit reference, the words are from Bruce – not mine. I strongly encourage everyone who wants to understand Open Source Software to read the excellent articles that Bruce has published at http://perens.com.

History of the Open Source Concept

Timeline

- February 1989, Richard Stallman released the GNU project version 1.0 under the GNU General Public License (GPL)
- June 1989, Bill Joy released a free version of UNIX under the University of California's Berkley Software Distribution (BSD) license.
- 2000, 17,000 OSS projects on SourceForge
- 2004, 74,000 OSS projects on SourceForge
- 2005, 105,000 OSS projects on SourceForge
- 2010, ???

Definition (OSD)

The distribution terms of open-source software must comply with the following criteria (http://www.opensource.org/docs/definition.php):

- 1. Free Redistribution
- 2. Source Code
- 3. Derived Works
- 4. Integrity of The Author's Source Code
- 5. No Discrimination Against Persons or Groups
- 6. No Discrimination Against Fields of Endeavor
- 7. Distribution of License
- 8. License Must Not Be Specific to a Product
- 9. License Must Not Restrict Other Software
- 10. License Must Be Technology-Neutral

Definition (OSD)

The distribution terms of open-source software must comply with the following criteria (http://www.opensource.org/docs/definition.php):

- 1. Free Redistribution
- 2. Source Code
- 3. Derived Works
- 4. Integrity of The Author's Source Code
- 5. No Discrimination Against Persons or Groups
- 6. No Discrimination Against Fields of Endeavor
- 7. Distribution of License
- 8. License Must Not Be Specific to a Product
- 9. License Must Not Restrict Other Software
- 10. License Must Be Technology-Neutral

Definition (OSD)

The distribution terms of open-source software must comply with the following criteria (http://www.opensource.org/docs/definition.php):

- 1. Free Redistribution
- 2. Source Code
- 3. Derived Works
- 4. Integrity of The Author's Source Code
- 5. No Discrimination Against Persons or Groups
- 6. No Discrimination Against Fields of Endeavor
- 7. Distribution of License
- 8. License Must Not Be Specific to a Product
- 9. License Must Not Restrict Other Software
- 10. License Must Be Technology-Neutral

Original Motivation : (A) Software Quality

The rationale behind the OSD:

- Eliminate the temptation to throw away many long-term gains in order to make a few short-term sales dollars
- Since our purpose is to make evolution easy, we require that modification be made easy
- For rapid evolution to happen, people need to be able to experiment with and redistribute modifications
- Users have a right to know who is responsible for the software they are using. Authors and maintainers have reciprocal right to know what they're being asked to support and protect their reputations.
- To get the maximum benefit from the process, the maximum diversity of persons and groups should be equally eligible to contribute to open sources

Original Motivation : (B) Prohibit License Traps

The rationale behind the OSD:

- To prohibit license traps that prevent open source from being used commercially
- To forbid closing up software by indirect means such as requiring a non-disclosure agreement
- The rights attached to the program must not depend on the program's being part of a particular software distribution
- Distributors of open-source software have the right to make their own choices about their own software
- Conformant licenses must allow for the possibility that (a) redistribution of the software will take place over non-Web channels that do not support click-wrapping of the download, and that (b) the covered code (or re-used portions of covered code) may run in a non-GUI environment that cannot support popup dialogues

The Rules In Plain English...

From [LR 2005]: *Open Source is built upon a foundation of IP law, particularly copyright law. OSS is owned by its authors, who license it to the public under generous terms … [but] The OSD is too confusing to focus readers on what really matters most…*

The key intention of the OSD is captured in five Open Source Principles:

Licensees are free to

- Use OSS for any purpose whatsoever.
- Make copies of OSS and to distribute them without payment or royalties to a licensor.
- Create derivative works of OSS and to distribute them without payment of royalties to a licensor.
- Access and use the source code of OSS.
- Combine OSS and other software.

Practical Reality 2005

- The key benefit of using Open Source components is reducing vendor lock-in, and the risk the vendor may go out of business or discontinue support for a product line
- To date the impact of OSS is felt mainly in the area of infrastructure software, and not in the business application space
- In the last five years many Open Source infrastructure software offerings have matured to the point of being rated best-in-class solutions by experienced software professionals
- In order to minimize TCO and maximize the life expectancy of software solutions, an understanding in which areas OSS can put to good use is becoming relevant

Note 1: The boundary between infrastructure (commodities) and value added software applications is slowly and continuously shifting. Hardly noticable, a bit like Global Warming…

Note 2: The term infrastructure or platform is relative, and depends on where you sit in the software supply/food chain

Distrust Simplistic Messages

- Open Source software is just another temporary fad generated by the IT [hype] industry that no one will speak of in five years time
  - Not true. OSS has been around for many years, and there is no indication that OSS will disapear. The distributed structure of OSS development and maintenance avoids the risks of a vendor going under and unsupported software. In a similar way, the distributed architecture of the Web is the key mechanism that makes the Web reasonably robust and resilient.
- Open Source software is the way of the future, and most business needs can be covered by OSS today
  - Not true. As we shall see during the course of this workshop, the economics of software development virtually guarantee a role for proprietary software, albeit a continuously changing role.

Essentials of OSS Licensing

Licensing Taxonomy

Open Source Licenses fall into four basic categories [LR 2005]:

Academic Licenses (BSD, …)

- Originally created by academic institutions. Allow the software to be used for whatever purpose whatsoever with no obligation on the part of the licensee to provide access to derivative works.

Reciprocal Licenses (GPL, … sometimes also called *viral licenses*)

- Allow the software to be used for whatever purpose whatsoever but require the licensee to distribute derivative works under the same license.

Standards Licenses (SISSL, …)

- Are used to ensure that industry standard software and documentation is available to all potential implementers of an industry standard. Some of these licenses require that differences from the standard be published as a reference implementation so that the standard can evolve.

Content Licenses (see www.creativecommons.org)

- Apply to copyrightable material other than software such as music, art, film, literature, etc. that the authors want to make available to the public domain.

Practical Reality

There are two public commons of free software:

Software licensed under Non-Viral Licences (Academic Licenses)

- Increasingly commercial software vendors donate infrastructure code to the Open Source community using non-viral licenses.
- It allows these vendors to benefit from the OSS development community, and it reduce vendor lock-in for their clients.
- It also motivates third parties to contribute, and to create value added products on a common OSS platform.
- A good example is IBM and the www.eclipse.org community.

Software licensed under Viral Licenses (Reciprocal Licenses)

- GNU and LINUX software is licensed under the GPL.
- The likely interpretation of the GPL by the courts will mean that derivative works are subject to the GPL's reciprocity (sometimes called "copyleft") provision, but collective works are not. [LR 2005]
- Note: Collective works are collections of independent works are assembled into a collective whole. Derivative works are based upon one ore more preexisting works, such as a translation…or any other form in which a work may be recast, transformed, or adapted. Still, it is easy to see that in the world of software, the boundary is fuzzy.

Certification of OSS Licenses

- Open Source Initiative (OSI, www.opensource.org) is a non-profit corporation dedicated to managing and promoting the Open Source Definition for the good of the community, specifically through the OSI Certified Open Source Software certification mark and program.
- OSI publishes approved open source licenses.

From www.opensource.org:

The basic idea behind open source is very simple: When programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing.

We in the open source community have learned that this rapid evolutionary process produces better software than the traditional closed model, in which only a very few programmers can see the source and everybody else must blindly use an opaque block of bits.

Open Source Initiative exists to make this case to the commercial world.

Basic Economics of Open Source Software

OSS Economics 2005

From Bruce Perens, Senior Research Scientist, Open Source Cyber Security Policy Research Institute, George Washington University (http://perens.com/Articles/Economic.html , the best explanation of OSS economics that I know of):

It's not immediately obvious how Open Source works economically. … the worst consequence of this lack of understanding is that many people don't understand how Open Source could be sustainable … if you look more deeply … it's easy to establish that Open Source is both sustainable and of tremendous benefit to the overall economy.

Open Source can be explained entirely within the context of conventional open-market economics. Indeed, it turns out that it has much stronger ties to the phenomenon of capitalism than you may have appreciated.

Outlook for LINUX

IDC Software Consulting (http://www.osdl.org/docs/linux_market_overview.pdf)

The Linux Marketplace – Moving From Niche to Mainstream

- *Packaged software is the fastest growing market segment within the Linux marketplace in terms of revenue, growing 44% annually to over $14 billion in 2008.*

Server market share predictions for Asia Pacific (excluding Japan) according to IDC (http://www.zdnetasia.com/insight/specialreports/0,39044853,39202771-4,00.htm)

- *For 2005, the overall spending for server systems is expected to grow at a rate of 6.2%. IDC pegs Microsoft Windows platform growth at 8.1%, Unix at 6.1%, and Linux at a none-too-modest 28%. In the longer term, compounded annual growth rates for the various server platforms are as follows: CAGR from 2003 to 2008 is 31% for Linux, 11% for Windows and 3% for Unix. By 2008, market share figures are pegged at Linux, 13%; Windows, 39.1%; and Unix, 38.1%.*

Availability of a large pool of MS certified people is a key factor in the increased adoption of Windows. In contrast, technically competent Linux professionals are still rare.

What Do These Numbers Mean?

The unusual acceptance and startling financial figures argue that Open Source must be the answer to some previously-unfulfilled need. Otherwise we would not have seen such wild, seemingly absurd phenomena:

- *The hobby project of a student in his twenties, Linux, takes over enterprise computing.*
- *IBM, the epitome of conservative business, de-emphasizes its billion-dollar "AIX" operating system in favor of a product developed by a loose coalition of programmers with no financial motive in common, upon whom no corporate directive can be binding, whose leader has no power but the respect of others.*
- *Microsoft faces its first serious competitor in a decade: programmers who give away their work.*

These events seem absurd: they certainly don't fit the common economic paradigm of technology production. A new economic phenomenon is operating, and to explain it we'll have to look more deeply into the economics of software production.

Grass Roots Level Motivation for OSS

Perhaps 90% of the software in any business is non-differentiating. Much of it is referred to as infrastructure, the base upon which differentiating technology is built. ... such things as operating systems, web servers, databases, Java application servers and other middle-ware, graphical user interface desktops, web browsers, email clients, spreadsheets, word processing, and presentation applications.

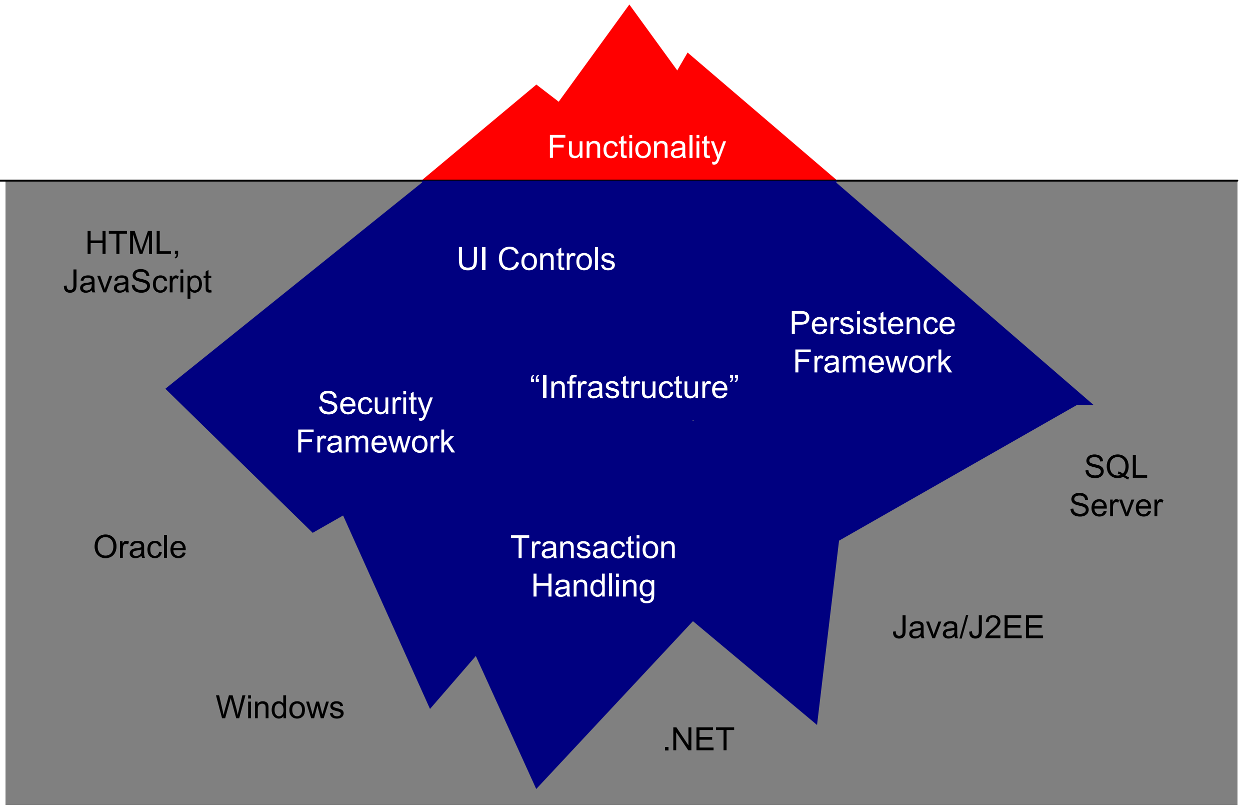
Good software developers are attracted to the idea of writing / contributing to OSS because it allows them to own the tools they build. Avoidance of employer lock-in: Why develop great tools, if they end up being owned by an employer?

The Software Iceberg

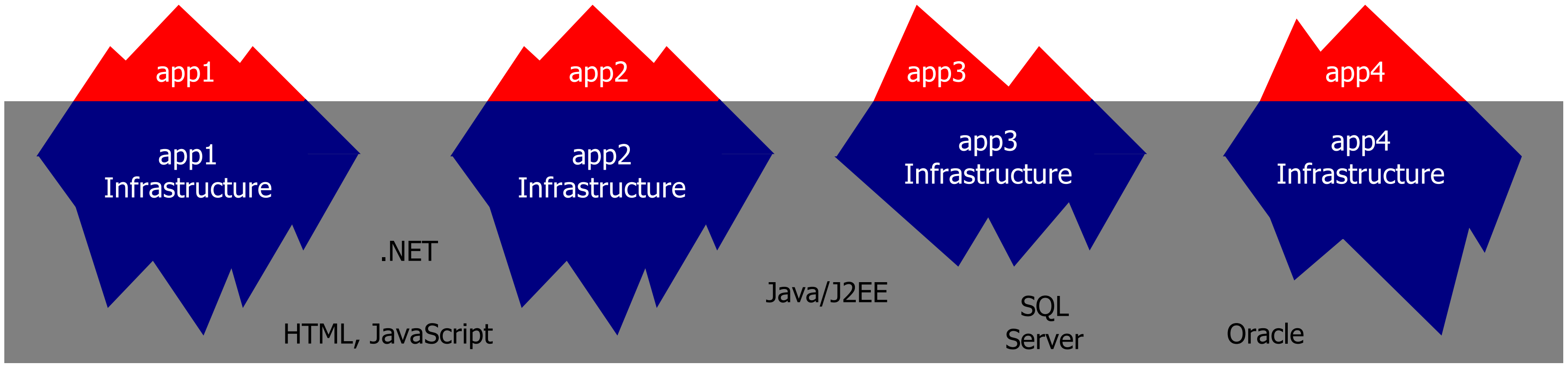
Functionality visible to Users
differentiating software

Infrastructure
non-differentiating software

Infrastructure development, maintenance and evolution should be shared



Infrastructure Redundancy



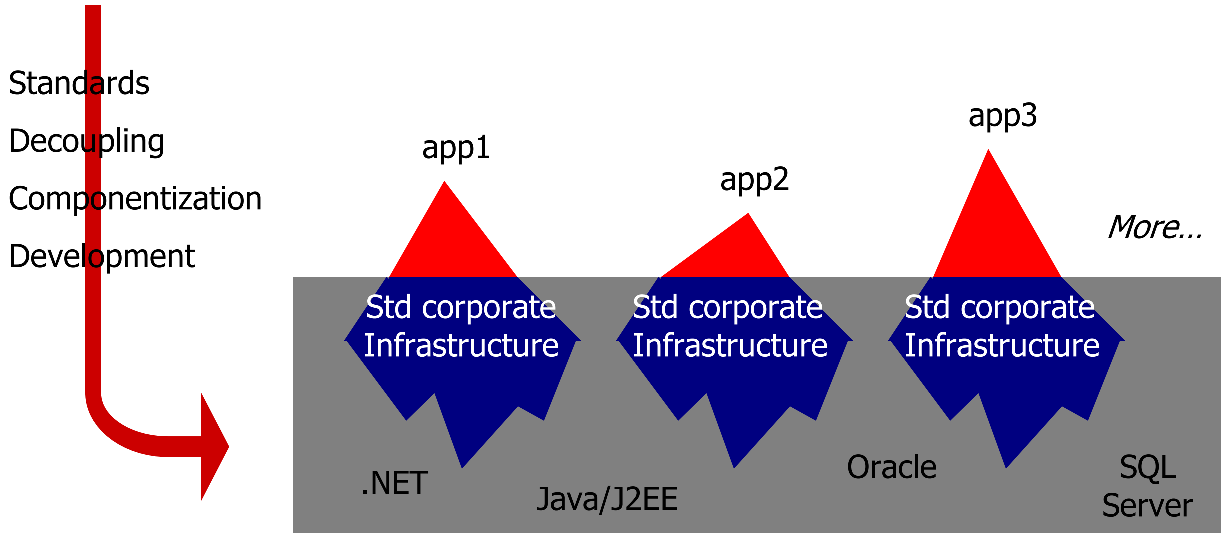
The infrastructure of package business software is not of concern and lies outside your control. However, the infrastructure of custom applications and also the infrastructure tools used to integrate packages into your system landscape is within your control.

A key motivation for establishing in-house software architecture standards is the reduction of infrastructure redundancy.

In a complex environment you will never achieve a "clean" target state, as you are always playing catch-up with technology churn.

OSS infrastructure is attractive because it offers a way of sharing the burden of infrastructure development not only across all your applications, but across a wider community, without creating any vendor lock-in.

OSS Centric Infrastructure Standardization



Reduction of duplication in infrastructure by incrementally migrating applications to a standard infrastructure.

The standardized infrastructure can consist of a mix of OSS and proprietary components.

Don't forget that software development is about communication and collaboration between people (Alistair Cockburn). The use of OSS infrastructure components is not only beneficial for your organization in terms of reduced vendor lock-in,

– It also enables your employees to acquire skills that are not tied to technology of a specific vendor.

– Easily transferable skills and knowledge: With a new employer, it is easier to bring on board useful OSS software tools than to argue the business case why the organization should purchase a specific proprietary infrastructure component.

Open Source > Buy > Build

Prospective software entrepreneurs are often asked: how are you going to be the Next Microsoft? And those who base a business upon Open Source are asked: how are you going to be the next Microsoft with Free software? [The question]reflects the fact that most people have been thinking about software from an extremely vendor-centric viewpoint.

[Only]Around 30% of the software that is written is sold as software. Most software is not sold at all. It is developed directly for its customer, by the customer's own employees or by consultants who bill for the service of software creation rather than for the end product.

For the successful introduction of OSS, take the following approach to infrastructure

- For each architectural concern that needs to be addressed: (1) Research OSS options (2) If no satifsfactory OSS components can be found, research COTS options (3) If no satisfactory COTS components can be found, ask yourself whether you have not fallen victim to "not invented here syndrome", and put too little effort into (1) and (2).
- Software developers love to write "cool code". And some may not be inclined to spend time evaluating COTS and OSS software. Provide incentives that make it attractive for your team to evaluate external infrastructure software. In particular don't expect your team to be embrace OSS if the team is measured by the lines of code produced or some equivalent measure.

Finer Points on the Economics of Software Development

(Strategic Assets, Commodities, and Liabilities)

Value-Based Classification of Software

The following classification scheme is a useful tool for planning investments in software:

- Strategic software assets—the heart of your business, assets that grow into an active human- and machine-usable knowledge base about your business and processes
- Non-strategic software assets—necessary infrastructure that is prone to technology churn and should be depreciated over two to three years
- Software liabilities—legacy that is a cost burden

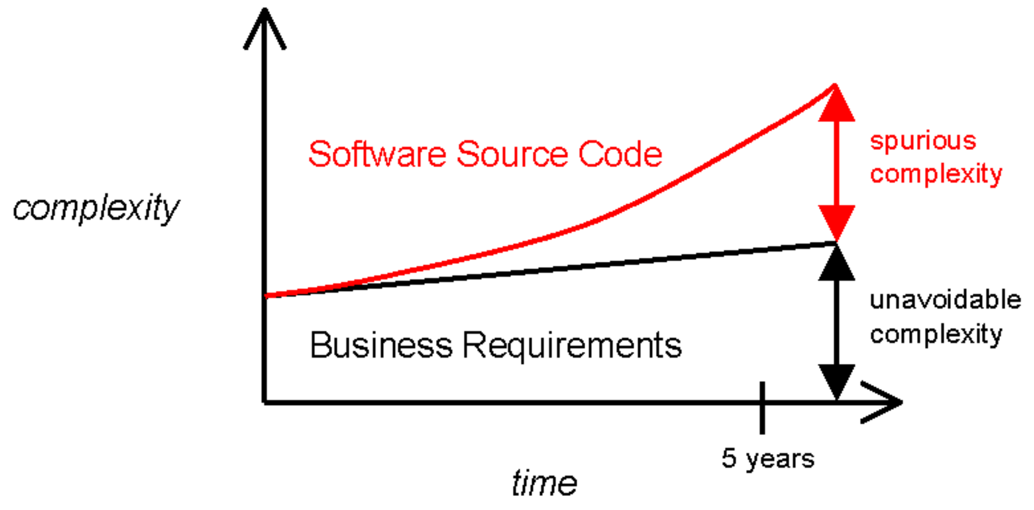
Classification of Software, easier said than done...

The previous slide made it sound easy, but:

- Can you clearly identify distinct parts/modules/sub-systems in your software system landscape?
  - Not just on Powerpoint slides, but in the software source code,
  - and in terms of deployable components.
  - Have you also got a firm grip on all the inter-dependencies between the components that make up your systems?
- Once you all dependencies have been identified and documented, are you able to group the components in such a way that
  - Some distinct non-overlapping subsets can be deployed independently and consititute meaningfull applications, and that
  - all other distinct non-overlapping subsets (when deployed in a particular sequence), add additional meaningfull applications to the mix?

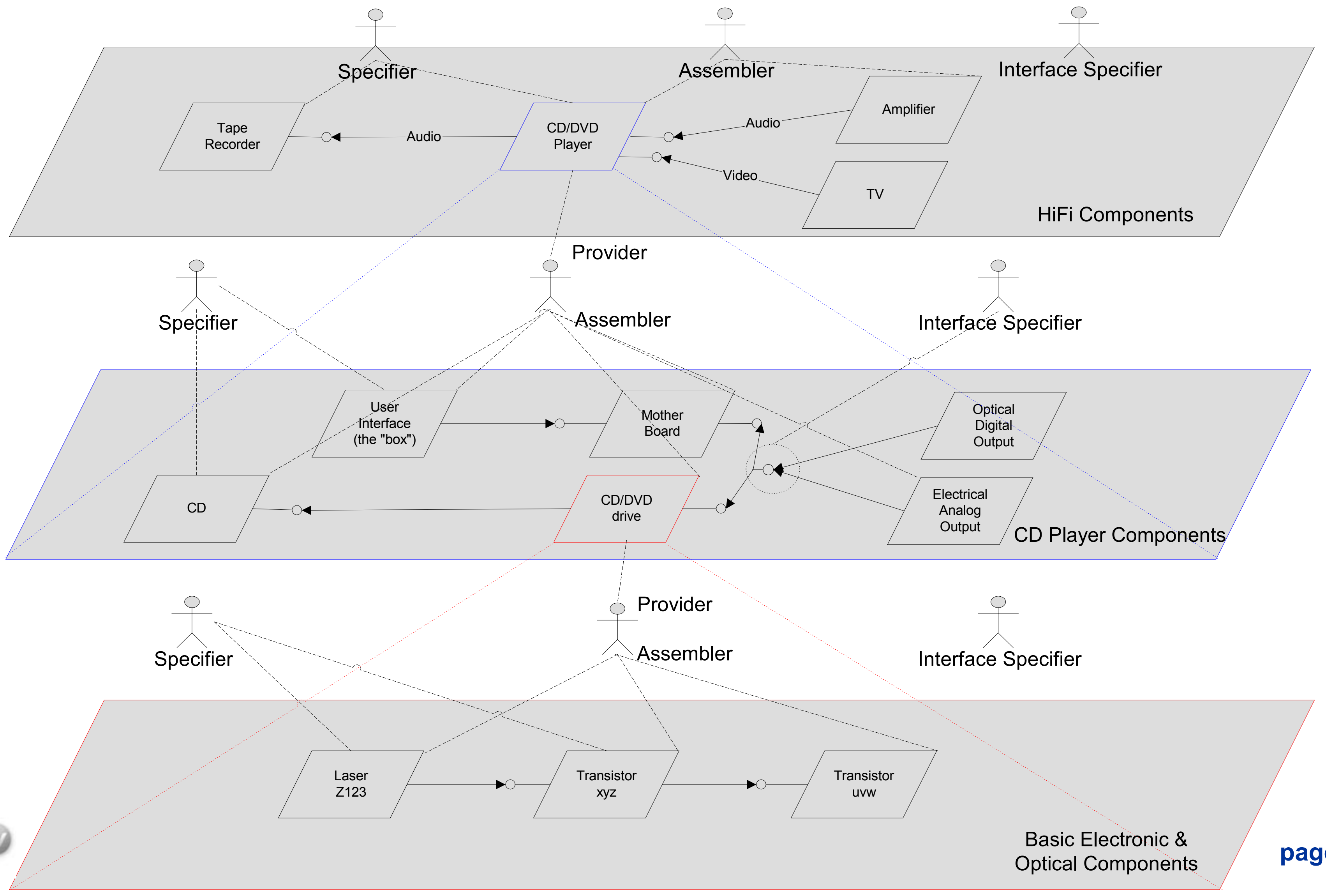
The Value of Modularization

Minimize spurious complexity, which is driven by the number of "random" inter-dependencies in your code base.

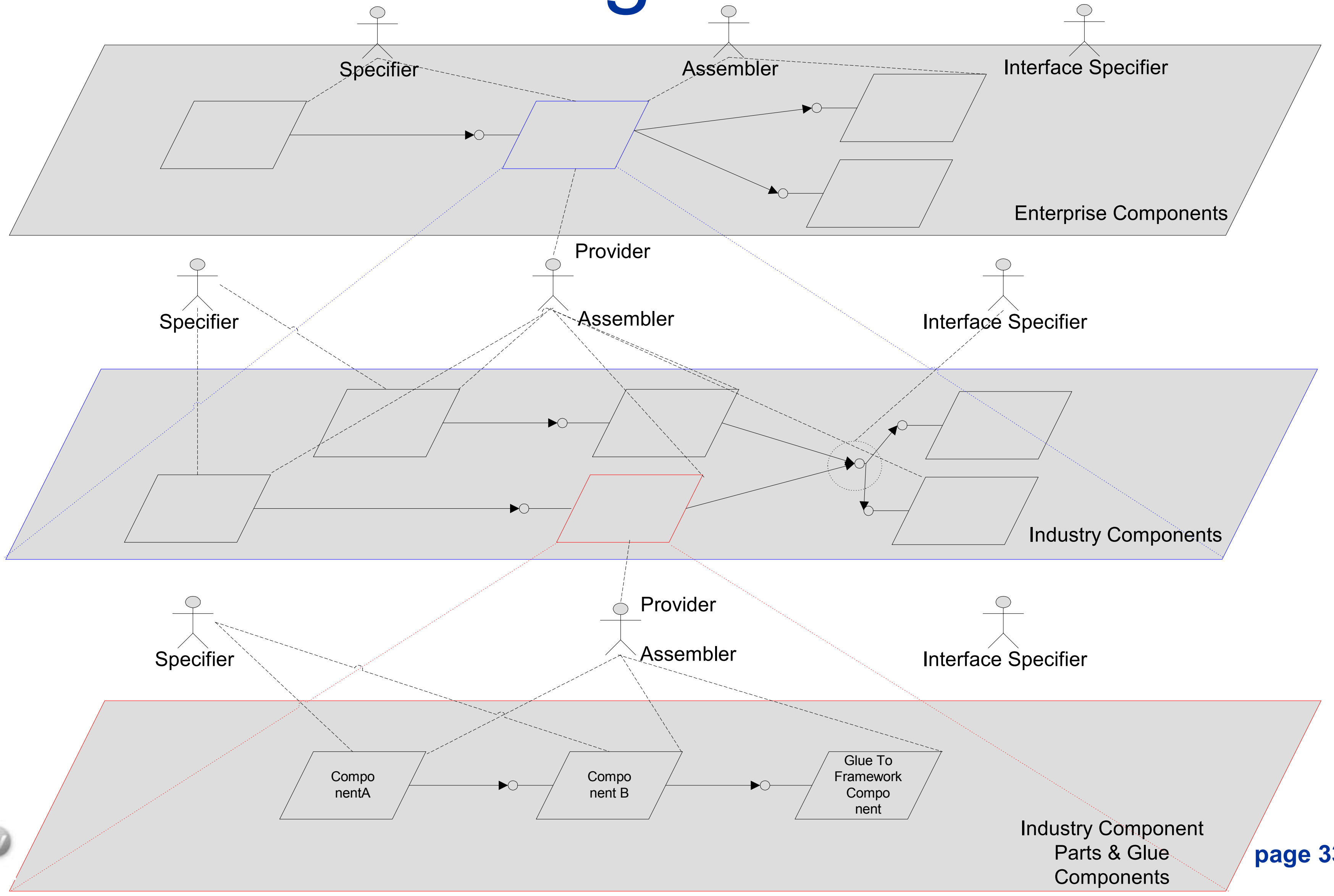


Note: A Component Architecture needs to define not only a "technology stack" (a set of infrastructure components), but also the allowable dependencies between various types of components.

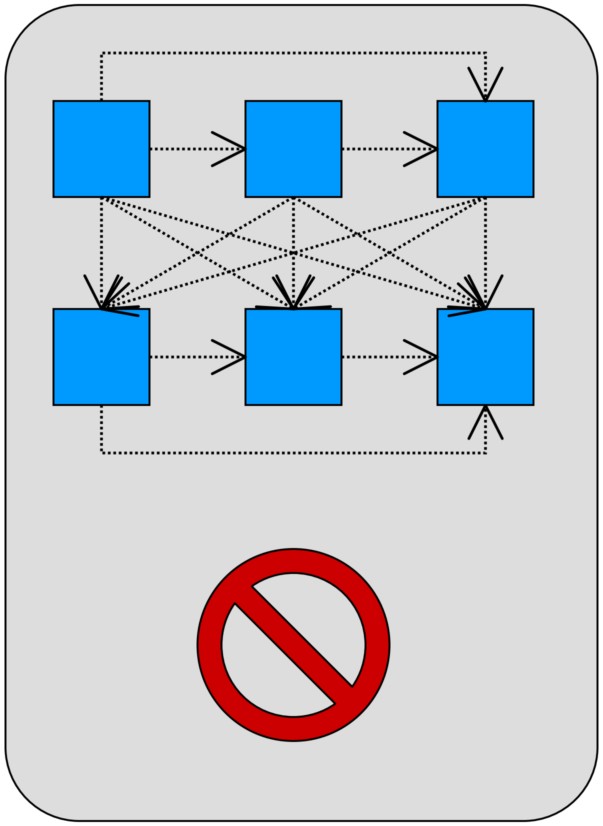
Component Architecture of Your HiFi System

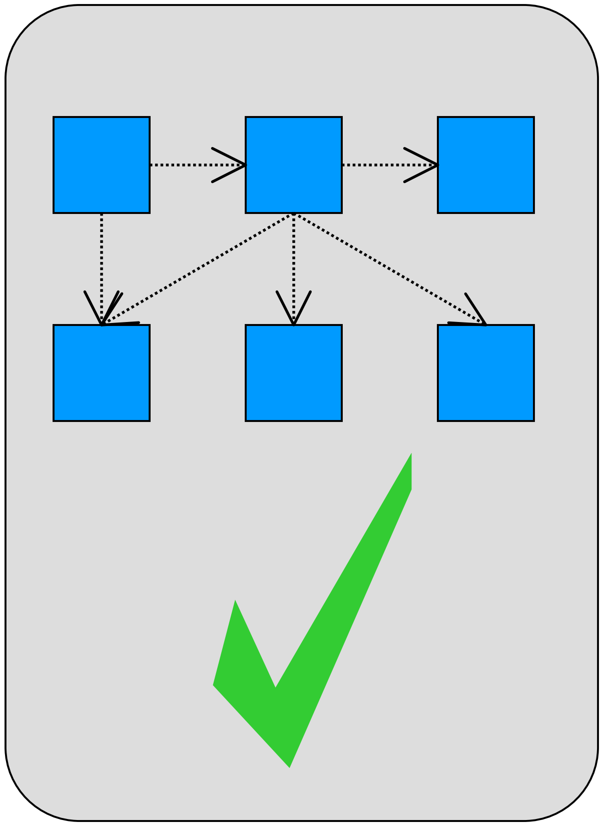


Component Architecture In Well-Designed Software



Dependency Management is required at all Levels of Abstraction





Dependency Management - the Raw Numbers

Careless construction of point-to-point interfaces is a deadly trap that looks harmless at first, and then “grows on you” with quadratically increasing impact

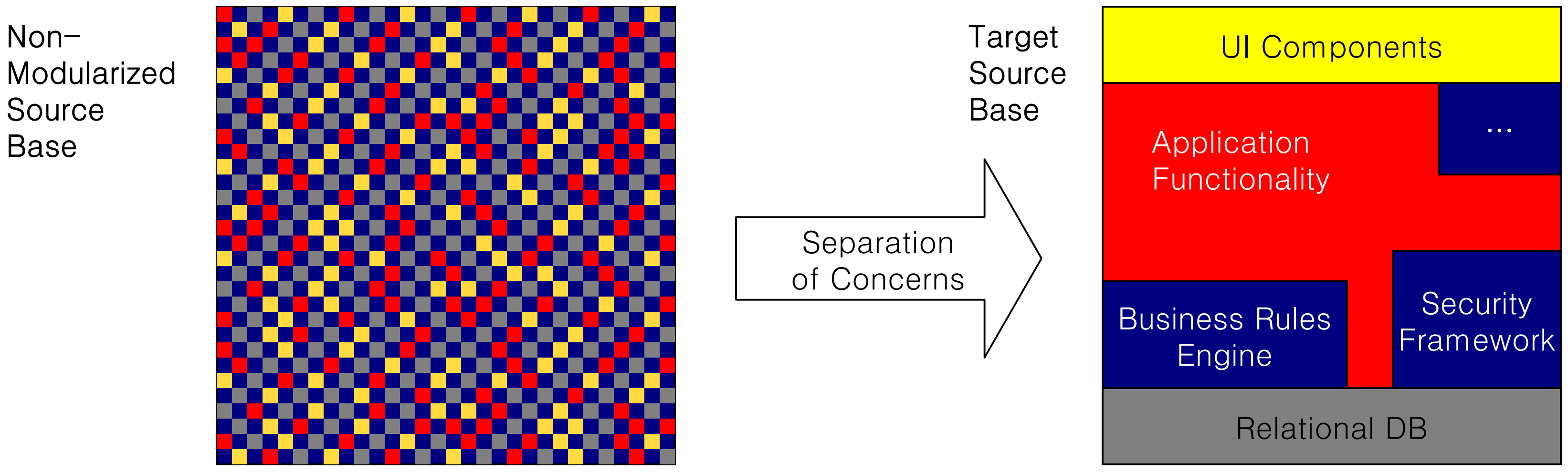
# of systems	# of potential point-to-point interfaces
2	1
4	6
8	28
16	120
32	496
64	2,016
128	8,128

Don’t get distracted by software architects who get lost in the micro design and only focus on low-level design patterns. The simplest preventative measure to curb spurious complexity without being prescriptive at the micro-level (software developers don’t tend to like that, and most like to have a degree of artistic freedom) is to religiously make use of a nested subsystem structure in the logical model of your system architecture. This strategy works regardless of the quality of design and implementation at the micro-level, as long as you deploy a mechanism that physically enforces this [one single] design rule.

# of systems	# of subsystems per system	# of potential dependencies within subsystem	total # of dependencies
1	100	4,950	4,950
10	10	45	495

The Maintenance Nightmare In Pictures

- Builds & Deployment are a major problem
- Everything depends on everything else
- Cutting corners doesn't work anymore: any perceived savings are eroded by the escalating costs of bug-fixes and new features



Code related to:

⬛ Relational DB	🟨 UI Components
🟦 Infrastructure	🟥 Application Functionality

- Evolving Functionality is "complex" as there are many dependencies
- Changing the DB platform would be close to a re-write
- A significant degree of complexity (= dev. costs) is avoidable by actively managing dependencies

How Can a Distributed OSS Project Produce Quality Software???

I hand over to Bruce Perens again:

*How do 50 people who haven't met work together to form a viable software product? Part of the reason this works so well is that **software is extremely modular by nature, and thus many people can work on different segments of the software, almost autonomously, if they can come to agreement about how the pieces fit together.** A good example of this is the Debian GNU/Linux Distribution. This system includes more than 16,000 software packages maintained by over 1000 volunteer developers in many nations around the world. When these packages are combined, the result is a reliable and well-integrated system. That system has supervised experiments while in orbit on the Space Shuttle, and has a user community second in size only to Red Hat (yes, it's bigger than Novell).*

A Darwinian selection mechanism is at work to ensure that only useful projects grow big, and the distributed nature of the development team ensures that only fairly well-modularized software sees the light of day. A co-located team has less “in-their-face” incentives to produce modularized code.

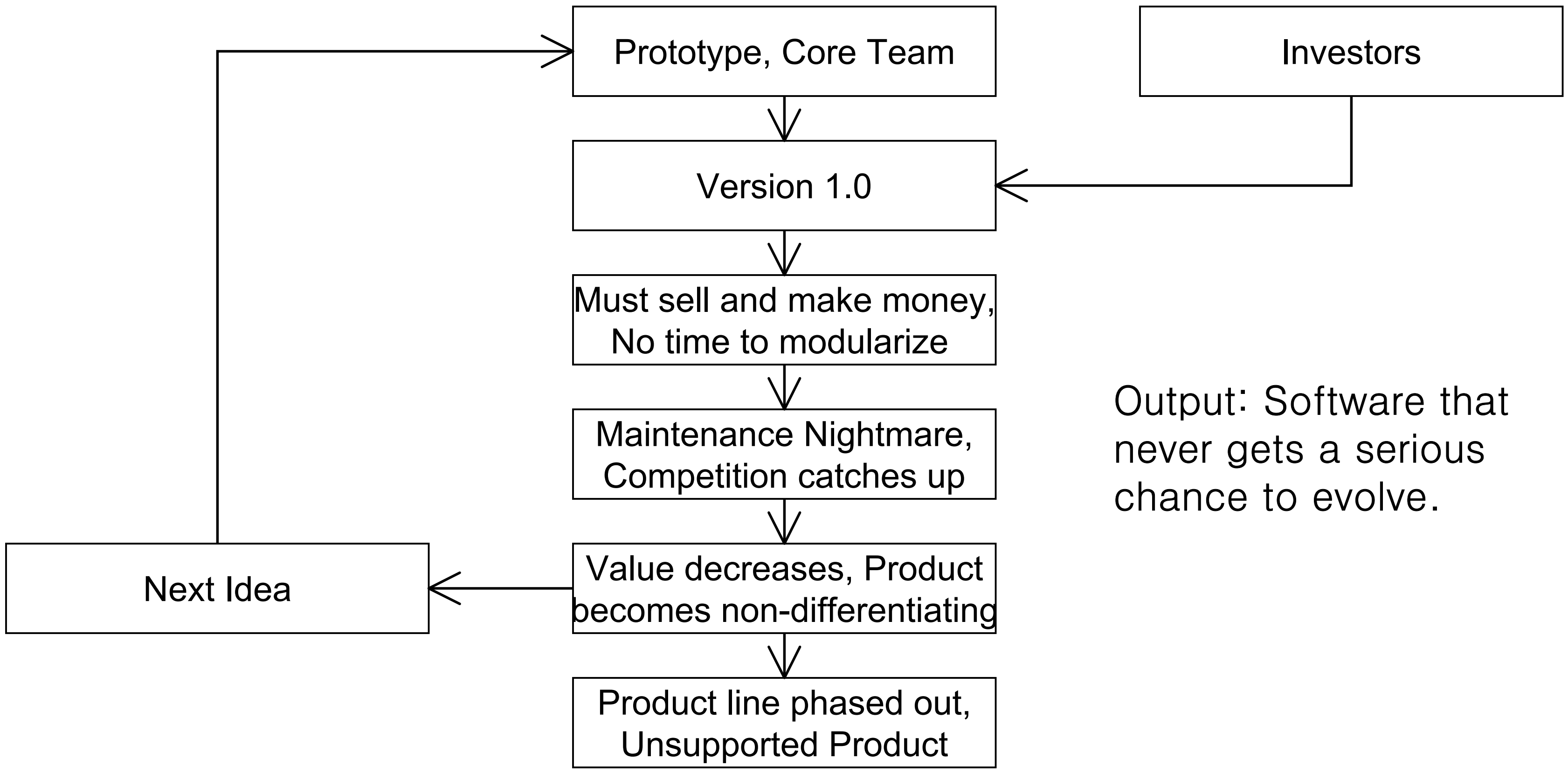
Examples of Darwinism at Work

Personal experience with OSS development:

- The Generative Model Transformer project (http://www.eclipse/gmt). A few years ago, together with a colleague, I started the Generative Model Transformer (GMT) project. Initially the project was not a spectacular success, and it has now undergone two major structural changes to date. The original code base of the GMT project has largely become irrelevant, as mature components such as the openArchitectureWare (oAW) tool have been integrated into GMT. oAW has attracted well over 30 developers.
- The Eclipse Modeling Framework (EMF) is one of the largest modeling projects on the Eclipse.org site. From the perspective of Model Driven Software Development the only interesting aspect of EMF is Ecore, the EMF implementation of the Object Management Group's Meta Object Facility (MOF) standard for meta modeling. Version 4 of oAW now allows the use of Ecore. The result is the best of both worlds: Ecore provides a meta meta model that conforms to the OMG's MOF industry standard, and oAW provides a powerful model driven template language and a high quality editor for that language.
- The Time Conscious Objects project (http://www.softmetaware.com/tco/overview.html) is still in its embryonic stage, waiting for a customer that is prepared to fund further development. The market determines whether the project is taken further.

Evolution in the Open Source gene pool is messy and the future not predictable. However the results of Open Source Darwinism are quite tangible and convincing.

Traditional SW Product Lifecycle


Prototype, Core Team
Investors
Version 1.0
Must sell and make money, No time to modularize
Maintenance Nightmare, Competition catches up
Value decreases, Product becomes non-differentiating
Next Idea
Product line phased out, Unsupported Product

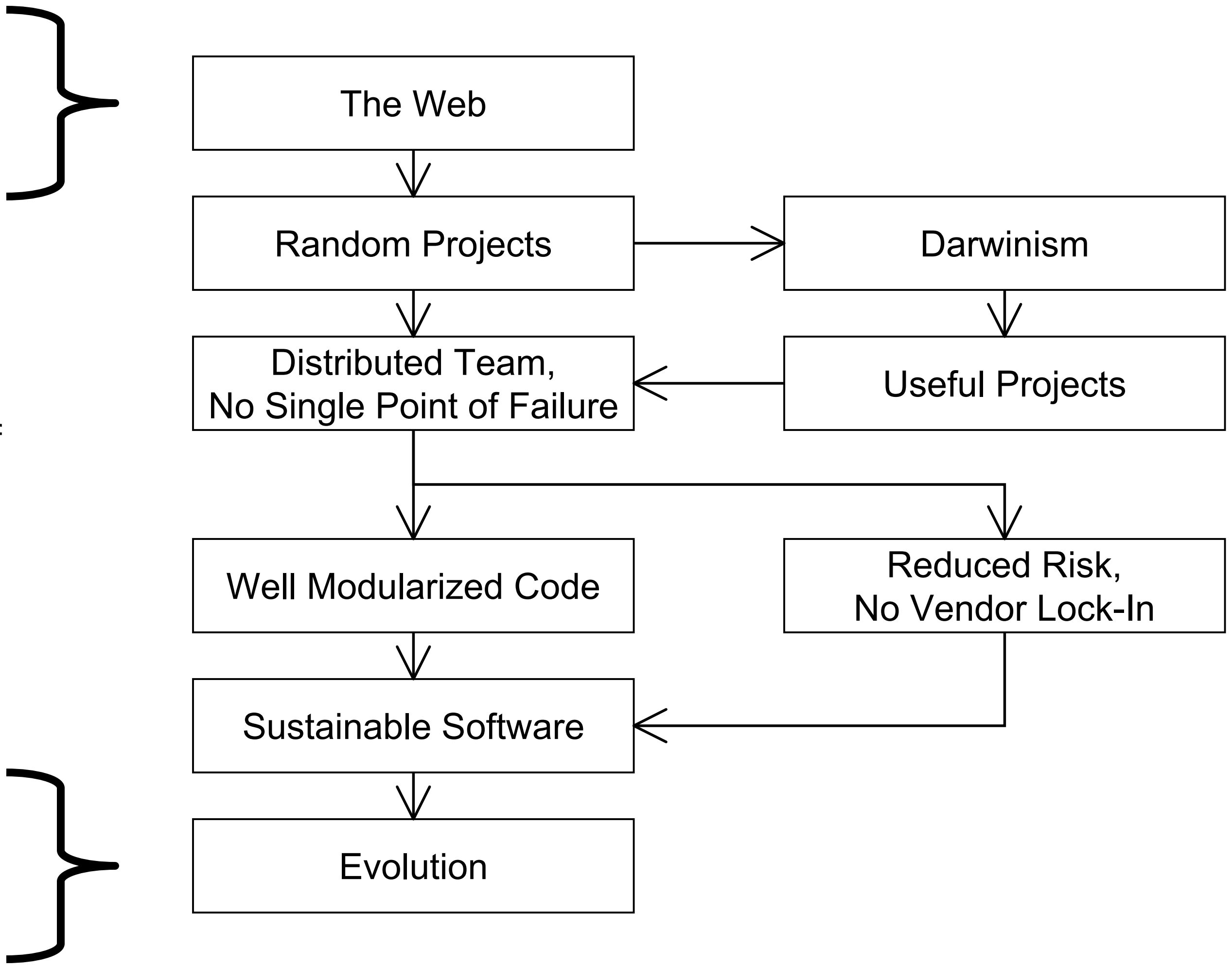
Output: Software that never gets a serious chance to evolve.

OSS Evolution

Primordial soup of resources that can interact via the web

Output: Software that allows one generation of software developers to build on the work of previous generations.

Incremental refinement and adaptation to changing environment



Open Source > Buy > Build

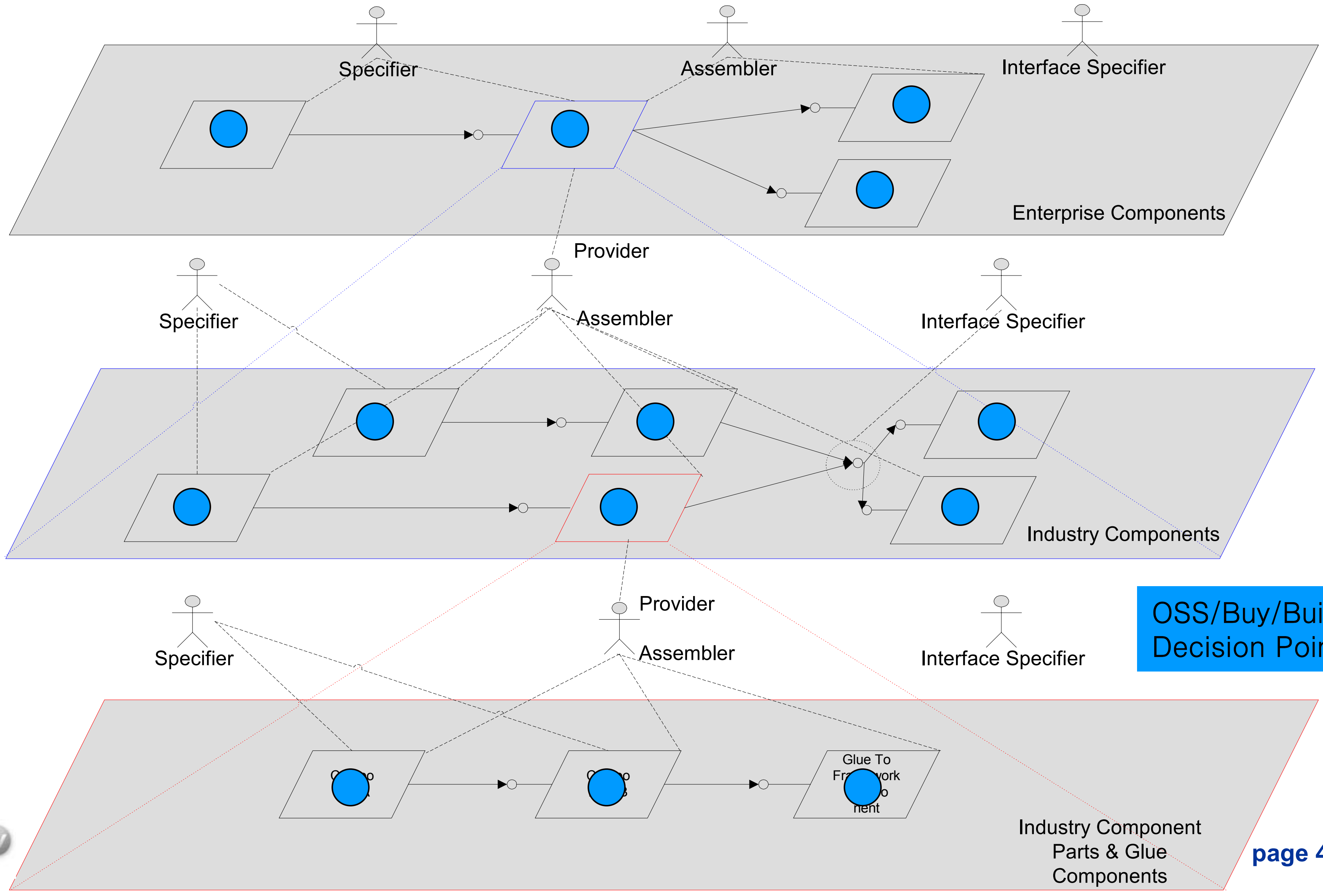
Build / Buy / Open Source decisions need to be made at a component level, and of course components can be nested.

- When buying application software, the vendor effectively dictates the technology stack, and no further decisions are required.
- The build option leads to further choices between using OSS infrastructure, buying third-party infrastructure, and using/building custom infrastructure.

Chances are that your custom applications are poorly modularized and heavily inter-dependent

- Most organizations, especially those that outsource customsoftware development, only have architectural standards and guidelines on paper.
- External service providers are very rarely held accountable for conforming to a well-defined component architecture. Mostly they are just held accountable for using a specific technology stack.

Decision Points



OSS/Buy/Build Decision Points

Lifetime Costs of Software Assets

For each software component

- consider the full cost of the initial purchase / development
- consider yearly license and maintenance costs
- condider cost of external resources involved in operation and maintenance
- consider lost opportunity costs of resources that could be doing something else
- consider transition costs that will be incurred when the component eventually needs to be replaced because of technological obsolecence

Off-the-shelf software is subject to the traditional software product development lifecycle. Therefore investments in COTS applications need to be depreciated over a realistically short period.

Evolving Your Software Portfolio

- Strategic software assets
  - The most economic model for these assets is in-house development.
  - If you outsource software development, it is highly recommended that you start holding suppliers accountable for conforming to well-defined component architecture standards. These standards need to be a key part of your Enterprise Architecture.
  - With the exception of niche areas where only one vendor offers a viable product, package software needs to be classified as non-strategic.
- Non-strategic software assets—necessary infrastructure
  - Some package software is very useful and the cost is often lower than in-house development. However most package software should be depreciated over two to three years.
  - OSS stands a higher chance of long-term survival than proprietary package software.
- Software liabilities—legacy that is a cost burden
  - Each day that they are kept on life support drains money that could be used to create value elsewhere. Need to be replaced ASAP, or simply phased out if replacement is not critical or perceived as too expensive. Early failure costs less than deferred failure.

The 30,000 Feet View: Getting the priorities right

1. Get serious about “componentization”

- Be realistic, take an incremental but disciplined approach. Start with a concrete application that needs to be developed or extended anyhow, so that you don’t introduce change for change’s sake.
- The average software developer is not worried about dependency management. Provide training and appropriate incentives.

2. Design a clean target component architecture for the selected application.

- Identify unmaintainable code as a liability that needs to be replaced. Have the courage to refactor.
- Research the web for appropriate OSS infrastructure components, and as necessary, adapt your component architecture to take advantage of OSS components.

3. Execute the project as your OSS pilot project.

Summary

Classification of software into strategic assets, non-strategic assets, and liabilities requires some homework in terms of "componentization" and dependency management.

Much of your custom software is likely to be poorly modularized, and therefore may easily turn out to be a liability when looking at the hard numbers of maintenance costs. Identification of sunk costs hurts, but sooner rather than later minimizes the overall costs.

The above is of critical importance when thinking about introducing OSS or other new infrastructure software, otherwise the new software will just contribute to further spurious complexity, and the project will not produce any measurable positive outcome.

Hence an OSS initiative needs to be tied in to a wider architectural initiative to reduce spurious complexity and to increase system maintainability.

The OSS Value Proposition in 2005

For Smaller Organizations with Little or No Custom Software

The **potential of OSS lies mainly in the operating system and desktop space**. The following considerations apply:

- Training your system administrators in the LINUX operating system, may be required.
- Determine which or your applications are capable of running on a LINUX server platform, and use the option to migrate servers to LINUX.
- Depending on your users, consider replacing at least some of the MS Office licenses with the use of OpenOffice, which is functionally nearly equivalent to MS Office.
- Replacing the desktop operating system with LINUX is a largely theoretical option.

For Organizations with Custom Software

- SourceForge hosts over 100,000 projects, most of which are targeted at software development professionals. Many of these projects represent dead wood, but this still leaves a staggering number of high quality OSS components that can be exploited.
- The real challenge is not the quality of OSS components in general, but knowing which OSS components are worthwhile to use.
- Especially in the area of software development tooling OSS is having a major impact. Arguably one of the best decisions that IBM made in the last five years was when IBM created Eclipse.org. Now there is a whole ecosystem of open-source projects that plug into the Eclipse platform. The number of downloads of the latest release: over 1,000,000 downloads within 60 days!

Open Standards

Open Standards - Principles

1. Availability

   Open Standards are available for all to read and implement.

2. Maximize End-User Choice

   Open Standards create a fair, competitive market for implementations of the standard.

3. No Royalty

   Open Standards are free for all to implement, with no royalty or fee.

4. No Discrimination

   Open Standards and the organizations that administer them do not favor one implementor over another. Certification organizations must provide a path for low and zero-cost implementations to be validated.

5. Extension or Subset

   Implementations of Open Standards may be extended, or offered in subset form.

6. Predatory Practices

   Open Standards may employ license terms that protect against subversion of the standard by embrace-and-extend tactics. The licenses attached to the standard may require the publication of reference information for extensions, and a license for all others to create, distribute, and sell software that is compatible with the extensions.

Open Standards - Practice

1. Availability

– The cost of a copy should not far exceed the cost of a college textbook.

– The best practice is for software reference platforms to be licensed in a way that is compatible with all forms of software licensing, both Free Software (Open Source) and proprietary.

2. Maximize End-User Choice

– They must allow a wide range of implementations, by businesses, academia, and public projects. They must support a range of pricing from very expensive to zero-price.

4. No Discrimination

– A standards organization that wishes to support itself through certification branding should establish a premium track and a low-cost or zero-cost track.

6. Predatory Practices

– The standards organization may wish to apply an agreement similar to the Sun Industry Standards Source License to the standard documentation and its accompanying reference implementation. This makes it possible for a standards organization to actively preserve interoperability without stifling innovation.

The Relationship between OSS and Open Standards

- Only in 2002 was an effective definition of open standards published by the World Wide Web Consortium (W3C) that was truly compatible with the Open Source concept!
- Standards are developed by industry consortia
- OSS provides an efficient mechanism to create working standards:
- *It [OSS] works better than consortia. Companies have poured millions into consortia to develop software standards. But they always go down in flames. And open-source projects win over and over again. Why? It's because open-source licensing makes things fair for all the partners. In the consortium projects, there's always the handshake with one hand and a dagger in the other.*

The Relationship between OSS and Open Standards

- Scot Peterson, HP: *Companies cooperate on standards and compete on implementations*
- Hence there is a tendency for standards to reflect lowest common denominator rather than state-of-the-art
- Development of industry standards is a very slow process. Examples
  - XMI (UML model interchange) and
  - QVT (model transformation standard for the OMGs Model Driven Architecture)
- Open Source implementations provide “substance” to standards that would otherwise remain theoretical

W3C Definition of Open Standards

http://www.w3.org/Consortium/Patent-Policy-20040205/#sec-Requirements

With respect to a Recommendation developed under this policy, a W3C Royalty-Free license shall mean a non-assignable, non-sublicensable license to make/use/sell/distribute implementations of the Recommendation that :

- Shall be available to all, worldwide, whether or not they are W3C Members
- May be limited to implementations of the Recommendation, and to what is required by the Recommendation
- May not be conditioned on payment of royalties, fees or other consideration
- May be suspended with respect to any licensee when licensor is sued by licensee for infringement of claims essential to implement any W3C Recommendation
- …

W3C Definition of Open Standards - Explanations

Implementations may be limited to implementations of the Recommendation

- Allows a field of use restriction that may limit the development of certain derivatives works

Implementations may not be conditioned on payment of royalties, fees …

- This is a key point that can be hard to explain to shareholders, i.e. that more value is created by distributing the IP freely and sharing in the public commons of free software
- Open Standards [and OSS] typically relate to commoditized infrastructure rather than business software. In other words, an open standardization initiative is always a sign that the industry is ready to commoditize an aspect of technology, in order to move on and reap greater rewards in application domains that critically depend on these commodities

Implementations may be suspended with respect to any licensee when licensor is sued by licensee for infringement of claims essential to implement any W3C Recommendation

- Similar to the way Open Source licensors are allowed to use their IP to defend against infrngement lawsuits by others

Discussion : OSS in Government Departments

Discussion Prompter ...

Government's use of Open Source is similar to the way that business approaches Open Source for its cost centers. However, government is expected to function for the benefit of the citizens and is not generally thought of as having profit-centers of its own. Rather, it provides services that enable economic and social activities.

Government contracting should not provide a commercial advantage to a particular vendor outside of the direct revenue from the products or services purchased. Government should especially not lock itself to a particular vendor after the contract term because of switching costs. It's poor policy for government to lock its vendors or citizens into use of a particular vendor's product for communicating with the government, as this would provide an inappropriate advantage to the vendor.

All vendors can make use of Open Source components with appropriate licensing, and thus can facilitate e-government to make use of Open Source for government-to-citizen, government-to-business and government-to-government interfaces.

Government carries out some activities solely for the public benefit, and can carry out Open Source development in this capacity. This is generally done through research funding.

A Realistic Approach to Enterprise Architecture

The Key Factor in an Enterprise Architecture : People

A well designed Enterprise Architecture covers the following areas:
People, Process, Business, and Technology
(see www.enterprise-architecture.info)

Trust is the foundation that fosters creativity in people. To create a trusting environment requires the following elements:

- Listening to employees and acting with urgency
- Appreciation for the work of employees, and recognition that people have an intrinsic value that's not related to the function they perform at work
- Executives taking responsibility for their actions instead of blaming others
- Executives communicate a shared vision and purpose among employees

When the word employee is substituted with contributor/user, and the word executive substituted with project leader/core contributor, then the above fairly accurately describes the atmosphere in successful OSS projects

Software Development

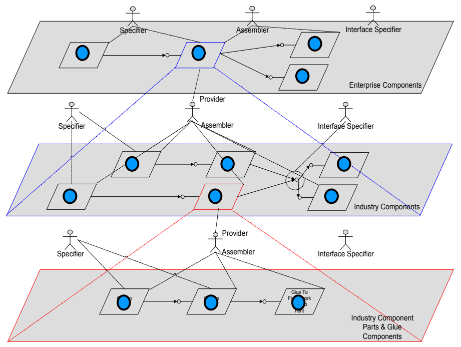
Alistair Cockburn (Humans and Technology, Cockburn and Associates)
http://alistair.cockburn.us/crystal/articles/cgm/cooperativegamemanifesto.html

Software development is a (series of) cooperative game(s), in which people use markers and props to inform, remind and inspire themselves and each other in getting to the next move in the game. The endpoint of the game is an operating software system; the residue of the game is a set of markers to inform and assist the players of the next game. The next game is the alteration or replacement of the system, or creation of a neighboring system.

To produce quality software on time and within budget it is necessary to create appropriate incentives for all parties involved with the objective of balancing the rights and responsibilities between end users, internal software professionals, and external contractors and service providers.

Aligning Business Objectives with IT

- Establish an appropriate balance of rights and responsibilities for all those involved – in IT and within the business units.
- Classify your software assets into strategic and non-strategic assets, and identify liabilities.
- For each software asset, make an informed OSS / Buy / Build decision

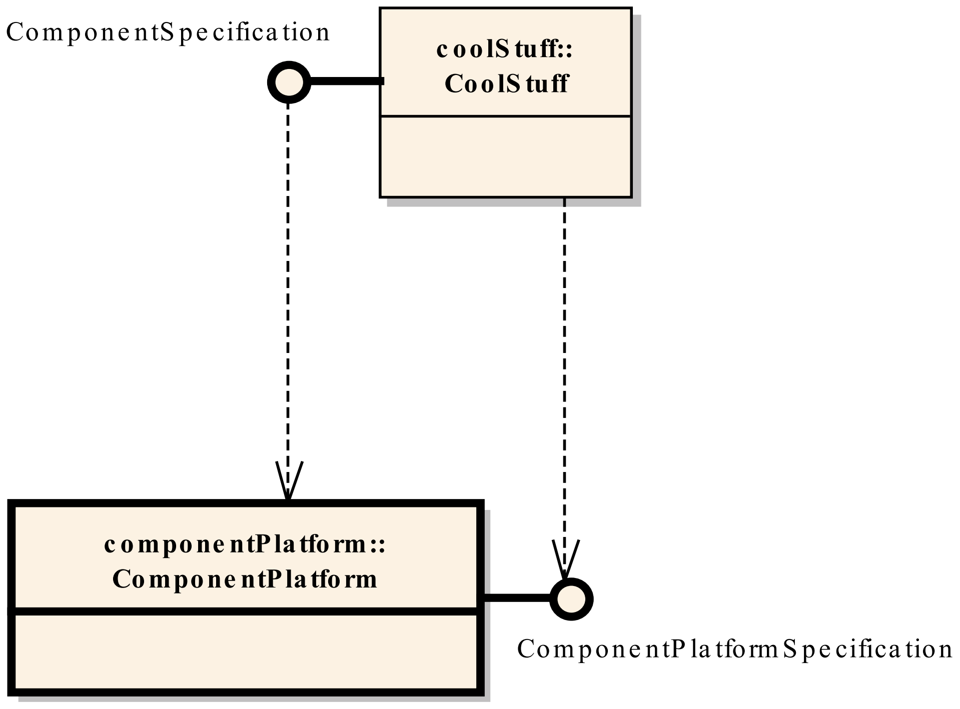


- Actively control and manage the dependencies between the components in your system landscape – enforce adherence to your EA
- Value the people who are contributing to IT initiatives. Motivation is largely the result of providing clear incentives and avoiding a discrepancy between official process and day-to-day reality

Technical Excursion : The Art of Modularization

The "façade" design pattern is a good example of a pattern that is obviously very useful, but is not specific enough to achieve active dependency management without further qualification.

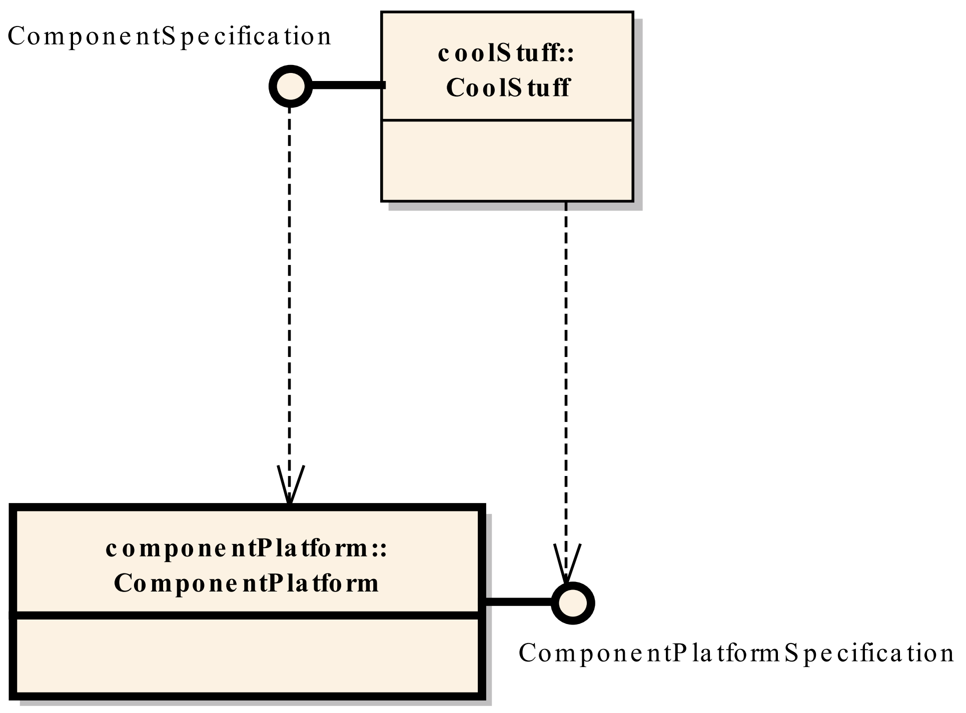
No subsystem lives in a vacuum. If a second subsystem communicates with the first via a façade, it depends not only on the façade, but also on the types exposed via the façade. If the second subsystem makes use of one of these types in its own façade, all of its clients end up depending on the first subsystem. The more subsystems end up depending on the types exposed in the façade of the first subsystem in the way just illustrated (i.e. not as direct dependents, but rather as "second cousins"), the less justification there is for keeping the type definitions in the first subsystem. However, none of the other subsystems would provide a more logical home either… This sets the scene for the concept of fully externalized interfaces.



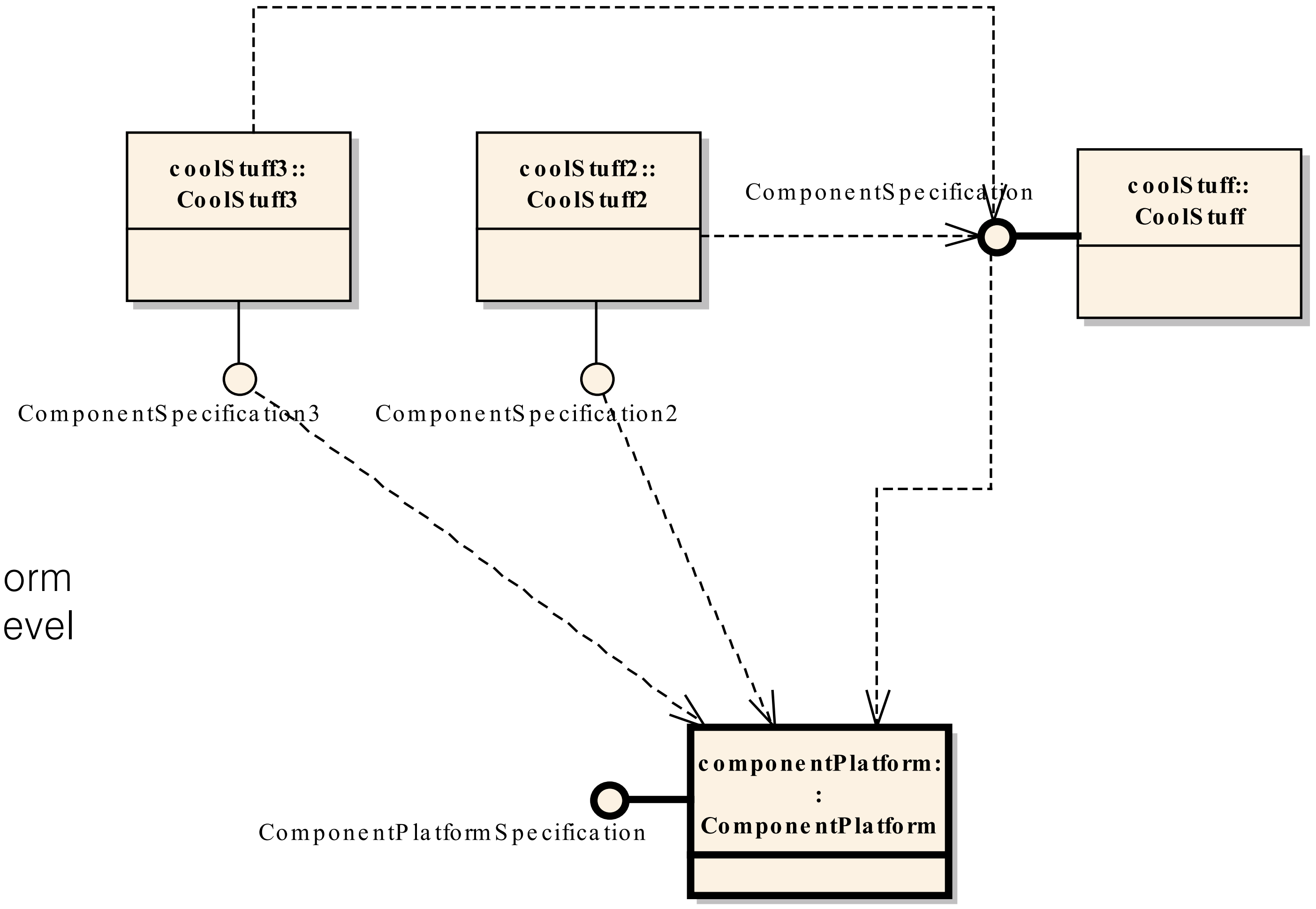
Fully Externalized Interfaces

If our component contains "cool stuff" that we want to promote for widespread use and reuse, then it is worthwhile to consider making certain pieces available in Open Source format using an appropriate OSS license.

More precisely, it may make sense to “open source” the Component Specification and the Component Platform (bold items in the diagram).

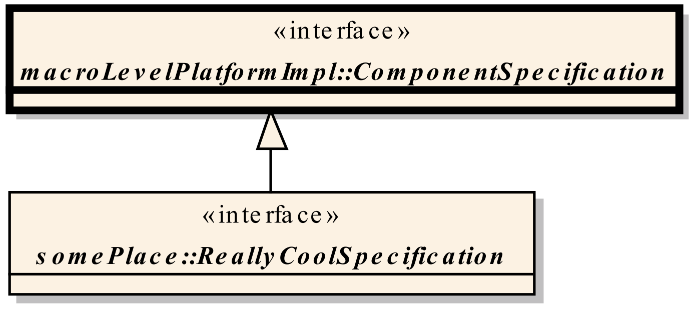


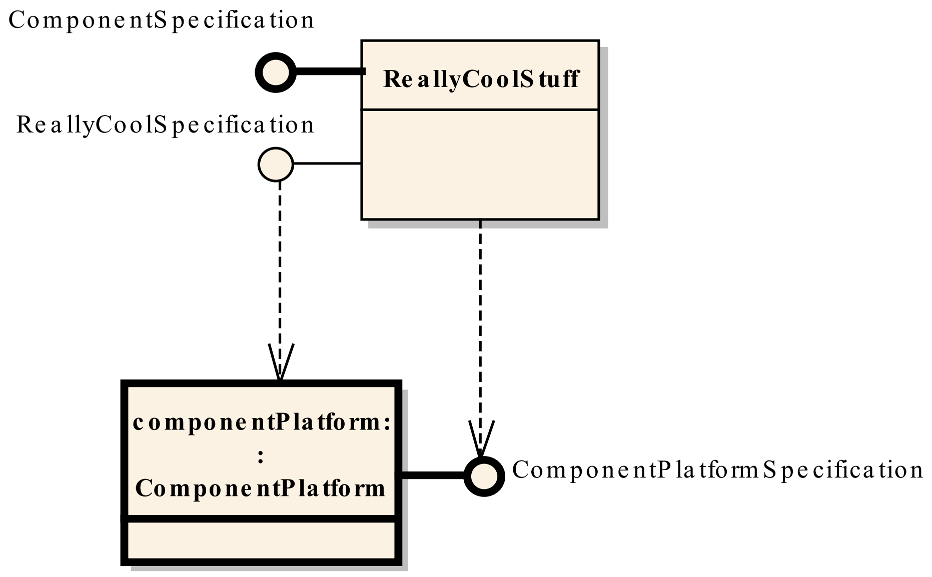
Standarizing the “Language” used between Components



Shared use of a Component Platform within a layer or level of abstraction.

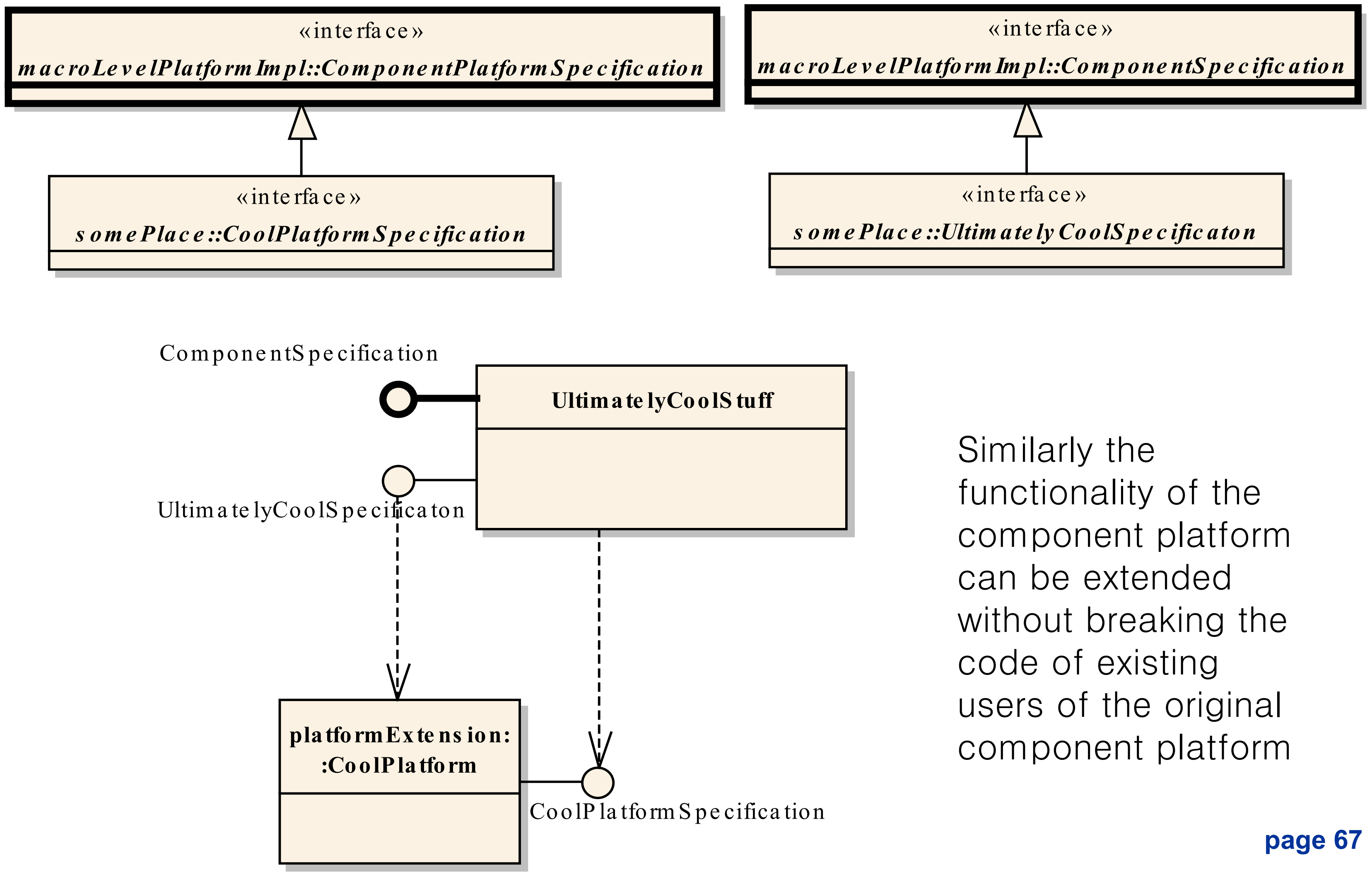
Component Extension/Evolution


«interface»
macroLevelPlatformImpl::ComponentSpecification
«interface»
somePlace::ReallyCoolSpecification


ComponentSpecification
ReallyCoolStuff
ReallyCoolSpecification
componentPlatform::ComponentPlatform
ComponentPlatformSpecification

Functionality of a component can be extended without breaking the code of existing users of the component

Component Platform Extension/Evolution


«interface»
macroLevelPlatformImpl::ComponentPlatformSpecification
«interface»
macroLevelPlatformImpl::ComponentSpecification
«interface»
somePlace::CoolPlatformSpecification
«interface»
somePlace::UltimatelyCoolSpecificaton
ComponentSpecification
UltimatelyCoolStuff
UltimatelyCoolSpecificaton
platformExtension: :CoolPlatform
CoolPlatformSpecification

Similarly the functionality of the component platform can be extended without breaking the code of existing users of the original component platform

And what about Service Oriented Architecture…?

SOA is a useful concept that needs to be factored into your enterprise architecture at the right level of abstraction.

It is useful to compare SOA to the Web. From a distance it looks chaotic, however as you get closer, you expect a meaningful structure at some point, i.e. when you have hit a home page of an organization. Web sites that lack structure are not very effective. Think of your enterprise architecture as a web site, and not as a chaotic, unstructured web.

- Use a top down approach to define how you interact with external parties. This leads to a design where services are explicitly traceable to core business processes.
- The service implementations can then pragmatically consist of a combination of new components and wrapped up legacy systems – the important point is not to compromise the external design of the services.
- It requires a good understanding of the business domain, and someone experienced in the skill of API lifting.

Using OSS Commodities

OSS From a User’s Perspective

A small survey of highly experienced software development professionals (all from different organizations) from three continents may not be statistically relevant, but nevertheless produces some interesting results:

- Only one participant had actively contributed to an OSS project
- Yet on average each participant had experience in using 8 OSS components, and had on average used each of these 8 components 4 times
- The average rating given to the OSS components was “best OSS component available on the market”, and 8 OSS components were rated to be “overall best component on the market for its intended purpose”.
- In total the experience of the 8 participants span 42 OSS components, and 263 instances of OSS component usage.
- All of the components that participants listed were infrastructure components, none were end user business applications

OSS Survey Keys

Usage Key	Only count the number of times you have embedded the component in different ways into your software. I.e. If you have written one wrapper for the component and have used the wrapper in three products or applications the count should be 1. If you have used the component in two different products or applications, exercising different use case [scenarios], then the count should be 2. Usage is capped at a value of 10, so that 10 effectively represents 10+.
	Please do include Linux etc. in the survey. But don't [go overboard] and list Open Source components distributed with Linux that you are not explictly using, or which Linux uses behind the scenes without you having an idea of how these components work and what the relevant APIs look like. In other words, list all the components where you have worked with the interface (whether a UI or API), and where you can therefore make a statement about quality/ease of use.

Rating Key	1=overall best component on the market for its intended purpose,
	2=best Open Source component on the market for its intended purpose,
	3=one of several useful similar components,
	4=does not quite meet my expectation but can be made to work,
	5=currently not worthwhile to consider

Experience Key	1=am a core contributor to the project,
	2=have contributed new features to the project,
	3=have used it to build an application/product,
	4=have evaluated it by exercising the functionality or writing test code to exercise the functionality,
	5=have not used it

Top Rating OSS

Top Rating OSS		Average		
Component	URL	Usage	Rating	Experience
eclipse.emf	www.eclipse.org/emf/	3.0	1.0	3.0
logging.apache	logging.apache.org	10.0	1.0	3.0
logging.apache.org.log4j	logging.apache.org/log4j/	1.0	1.0	3.0
lowagie.itext	www.lowagie.com/iText/	3.0	1.0	3.0
lucene.apache	lucene.apache.org	2.0	1.0	3.0
openArchitectureWare	www.openarchitectureware.org	10.0	1.0	1.0
junit	www.junit.org	6.0	1.4	3.2
jakarta.apache.tomcat	jakarta.apache.org/tomcat/	2.0	1.5	3.0
jboss	www.jboss.org	5.5	1.5	3.5
linux	www.linux.org	1.5	1.5	3.0
ant.apache	ant.apache.org	8.0	1.8	3.0
eclipse	www.eclipse.org	6.6	1.8	3.0
antlr	www.antlr.org	2.0	2.0	3.0
apache.axis	ws.apache.org/axis/	1.0	2.0	3.0
aspectj	www.eclipse.org/aspectj	1.0	2.0	3.0

Most Used OSS

Most Used OSS		Average		
Component	URL	Usage	Rating	Experience
logging.apache	logging.apache.org	10.0	1.0	3.0
openArchitectureWare	www.openarchitectureware.org	10.0	1.0	1.0
jakarta.apache.jmeter	jakarta.apache.org/jmeter/index.html	10.0	2.0	3.0
jedit	www.jedit.org	10.0	2.0	3.0
netbeans	www.netbeans.org	10.0	2.0	3.0
squirrel sql client	squirrel-sql.sourceforge.net	10.0	2.0	3.0
ant.apache	ant.apache.org	8.0	1.8	3.0
apache	httpd.apache.org	7.7	2.3	3.0
eclipse	www.eclipse.org	6.6	1.8	3.0
cvs	www.cvshome.org	6.3	2.5	3.0
junit	www.junit.org	6.0	1.4	3.2
jboss	www.jboss.org	5.5	1.5	3.5
postgresssql	www.postgresql.org	5.0	2.0	3.0
mysql	www.mysql.com	3.5	2.0	3.0
eclipse.emf	www.eclipse.org/emf/	3.0	1.0	3.0

Note that the LINUX is not part of the list of most used
15 out of 42 OSS projects

OSS Infrastructure in the Australian Government

(1) It turns out that the Australian Bureau of Statistics uses a subset of the top 15 components from the survey, and has around 18 months experience in developing software with OSS infrastructure.

- Low cost of initial participation has been a key driver
- The Eclipse platform is successfully used as a software development and tool integration platform: user interfaces are being built using Eclipse components, and custom tools are developed as Eclipse plug-ins

Several components from the Apache project are being used

As a result of the success to date, the use of further OSS components is being investigated

(2) The Department of Veterans Affairs is/was considering replacing MS Office with OSS office software, but is encountering some difficulties.

Building OSS Assets

The Cost of Building OSS

The cost-of-participation in mature Open Source projects is very different from the costs of retail or in-house and contract development. The major expense is the time-cost of employee participation. This figure is a combination of the personnel cost of software evaluation, the personnel or contractor resources spent to adapt existing Open Source to customer needs and to support the Open Source for internal users, and the possibility that time will be invested into software that is eventually replaced due to a failure to track customer needs. The maximum cost for Open Source would come when there is no community other than the customer: this would be similar to the cost of contract or in-house development, in which one customer supports the entire expense. The actual cost will be lower depending on the number of active participants and the work required. The lost investment is generally personnel time.

Taking this into account, the Open Source paradigm yields an economic efficiency at least as great as the in-house and contract development paradigm, and much greater than the retail paradigm.

OSS - “The” Paradigm for Infrastructure Software

If you have determined that you need infrastructure software that is not available in the form of existing OSS, and where commercial proprietary options are either also non-existent, or limited to very few vendors with high price tags, then you may want to consider building and releasing appropriate componentry under an Open Source license.

- You don’t lose anything by donating non-differentiating software to the public domain
- Instead you potentially provide a seed that gets picked up by other interested parties who contribute valuable features from which you – and the software community as a whole – can benefit.
- If, at some point in time, a commercial software vendor takes an interest, and develops useful add-on functionality, the result is positive rather than negative: the OSS project community remains in control of the OSS foundation, and the vendor won’t bite the hand that feeds them.

OSS as a Quality/Standarization Driver

Making a Component Specification and the corresponding Component Platform including its specification Open Source, promotes competition for cost efficient implementations. The first step might be a competitor who comes up with a better implementation of your "cool stuff", i.e. it may perform better, require less memory, but adheres to the same specification. In particular if your "cool stuff" has been successful (others have built on it), there will be a strong incentive for the competitor to support the interface of the original "cool stuff" component.

However, why would a commercial software vendor be interested in opening up commercially successful software in that way? Think about the following:

- Making the component specifications available in Open Source form sends a strong message to the market: we have this "cool stuff", and we want you to use it without needing to worry about vendor lock-in.
- Price is not everything. If the potential market is large, a quick expansion and development of the market may lead to more revenue than a high price and a completely closed architecture.
- The timing in making the relevant source code Open Source can be used as a tool to manage revenue. It makes sense to have a fairly well featured component/product when “open sourcing” the specification. The competition will take a while to produce a cheaper clone, which provides a window of time for recouping the investment into the first release. In case the original implementation is of high quality, the competition might never catch up. Otherwise, the original developer can use a competitor's improvement as a stepping stone forward.

Developing a Practical Roadmap for OSS Adoption

In what areas of your EA do you see the biggest potential for OSS?

1. Get serious about "componentization"

- Be realistic, take an incremental but disciplined approach
  …
- Provide training and appropriate incentives.
  …

2. Design a clean target component architecture for the selected application.

- Identify unmaintainable code as a liability
  …
- Research the web for appropriate OSS infrastructure components, and as necessary, adapt your component architecture to take advantage of OSS components.
  …

3. Execute the project as your OSS pilot project!

Open Discussion
&
Wrap Up

References and Further Reading

Open Source Foundations

[BP]	Articles by Bruce Perens (http://perens.com), in particular: http://perens.com/Articles/Economic.html
[DOS 1999]	Editors Chris DiBona, Sam Ockman, Mark Stone, Open Sources, Voices from the Open Source Revolution, 1999, O'Reilly, ISBN 1-56592-582-3
[ER 1999]	Eric S. Raymond, The Cathedral & the Bazaar, 1999, O'Reilly, ISBN 0-596-00108-8
[MF 2003]	Martin Fink, The Business and Economics of Linux and Open Source, 2003, Prentice Hall, ISBN 0-13-047677-3
[OSI]	Open Source Initiative (www.opensource.org)

Open Source Licensing, Copyright & IP Law

[AL 2004]	Andrew M. St. Laurent, Open Source & Free Software Licensing, 2004, O'Reilly, ISBN 0-596-00596-00581-4
[BP]	Articles by Bruce Perens (http://perens.com), in particular: http://perens.com/Articles/PatentFarming.html
[CC]	Creative Commons (http://creativecommons.org)
[LL 2004]	Lawrence Lessig, Free Culture, 2004, The Penguin Press, ISBN 1-59420-006-8
[LL 2001]	Lawrence Lessig, The Future of Ideas, 2001, Vintage Books, ISBN 0-375-72644-6
[LR 2005]	Lawrence Rosen, Open Source Licensing, Software Freedom and Intellectual Property Law, 2005, Prentice Hall, ISBN 0-13-148787-6

Managing Complexity

[AB 2000]	Alan W. Brown,<u>Large-Scale Component-Based Development</u>, 2000, Prentice Hall, ISBN 0-13-088720-X
[ABB 2002]	Atkinson, C., Bayer, J., Bunse, C., et al, <u>Component-based Product Line Engineering with UML</u>, 2002,Addison-Wesley, ISBN 0-201-73791-4
[JB 2003]	Jorn Bettin, <u>Complexity & Dependency Management</u>, http://www.softmetaware.com/complexity-and-dependency-management.pdf
[JB 2004]	Jorn Bettin, <u>Model-Driven Software Development: An emerging paradigm for industrialized software asset development</u>, http://www.softmetaware.com/mdsd-and-isad.pdf
[LLBR 2005]	Editors: L. Liu & B. Roussev, <u>Management of the Object-Oriented Software Development Process, chapter Managing Complexity with MDSD</u> (Jorn Bettin), 2005, Idea Group Publishers, ISBN 1-59140-605-6
[Meyer 1997]	Bertrand Meyer, <u>Object-oriented software construction</u>, 1997, Prentice Hall, ISBN 0-13-6291554
[MM 2002]	Richard Mitchell, Jim McKim, <u>Design by Contract by Example</u>, 2002, Addison-Wesley, ISBN 0-201-63460-0

Thank You!

Jorn Bettin

jorn.bettin@softmetaware.com
www.softmetaware.com

Mobile +41 79 543 3767
Skype Jorn_Bettin

Enabling Your Software To Evolve!