

# Adapting Extreme Programming to Research, Development and Production Environments

Gil Broza

545 Douglas Ave.  
Toronto, ON M5M 1H7 Canada  
gilbroza@hotmail.com

**Abstract.** Affinium Pharmaceuticals engages in early-stage pharmaceutical R&D and molecular biology production processes for internal and external programs. This business requires significant informatics support in terms of small- and large-scale software, tool integration and data management. Obtaining suitable software is difficult due to customer diversity, rapidly evolving unique needs, vendor offering and high costs. Adapting the XP approach and practices for this situation, Affinium's Informatics group has successfully developed in-house software that has kept up with the science. I describe notable accomplishments, and lessons learned along the way. I propose that a small in-house group of domain-aware developers, using a customized version of XP, would achieve better results than external providers, despite limited access to resources. In closing, I suggest that this structure and methodology are generally applicable to dynamic research, development and production environments.

## 1 Introduction

Affinium Pharmaceuticals is a structure-guided drug company focused on the discovery of novel anti-infective medicines. Its undertakings thus encompass scientific research, development and production. The drug discovery world is navigated using high-throughput processes for protein production, structure determination, synthesis of chemical matter and myriad assays for drug viability. R&D is responsible for rapidly obtaining drug candidates via proprietary automation, workflows and protocols.

Drug discovery is an extremely expensive and time-consuming pursuit. On average, developing a novel drug costs over US\$800-million and takes 15-years from initial research to marketed product [5]. To stay ahead and reach the market faster – and make the most of patent protection – companies must constantly innovate. Affinium's chosen path is that of utilizing molecular structure, through structural biology, computational chemistry, cheminformatics and bioinformatics. These pursuits are well known for the sheer volume of data they produce and manipulate. Sometimes reaching terabytes per month, this data may yield immense benefits when mined.

Affinium's line of business requires diverse software for lab and data management, maintaining and accessing scientific databases, streamlining workflow, and operating and integrating instruments. Software is purchased from vendors, outsourced to consultants, or built internally. In the next section I discuss the downsides of purchasing

commercial software, and several of the difficulties around planning and engineering useful, affordable solutions, whether done internally or outsourced. I then show how the Informatics group at Affinium has adapted XP to address these problems in building our own software or integrating 3<sup>rd</sup>-party software.

## 2 The Problem: Obtaining Appropriate Software

The dynamic nature of the industry poses challenges however a company acquires its software. Systems, which cost a lot of time, money and effort to obtain and integrate, often lose out to innovation and the fast pace of scientific discovery and automation advancement. New business initiatives, or changing priorities in a difficult economic landscape, may trigger obsolescence even faster. Gentler changes, such as new business collaborations or discovery programs, often introduce or modify requirements.

There exist several established vendors of informatics software for biotechnology and pharmaceuticals, but companies in these fields need to do more than choose among them. Given the highly diverse nature of activities in drug discovery, few vendors provide solutions that span the entire spectrum. Some needs are so special that no commercial product addresses them effectively. Lastly, vendors may not be quick, cheap or still around to respond to the changing nature of the business (e.g. [4]).

Data and process integration are vital to a company's scientific advancement regardless of its size or age. Data is produced and processes are developed in highly specialized departments of different disciplines, such as molecular biology, structural biology and computational chemistry. However, comprehensive data analysis and streamlined discovery operations require integration, posing serious challenges for software development. Achieving useful integration, whether between vendors' products or custom-built products, is time-consuming and expensive. Cost/benefit analyses lead many companies to opt for a mixed strategy.

A significant downside to eschewing vendors is that a desired system may take many months to enter production. Meanwhile, a lot of money is spent (and a lot more if the project is outsourced), and the requirements mutate. Unformatted or unstructured 'legacy data' piles up, which is difficult to use and later import. The scientist customers are often content getting by on their own, for instance by downloading free tools, writing Perl scripts and using Microsoft Excel. Thus, spending money to build software or database systems requires serious justification.

Developer skill-sets are another concern for companies building their own software. The subject matter is difficult to master due to its scientific breadth and specificity, so experts in both software engineering and biology or chemistry are few. Yet, successful developers must have a fairly good exposure to it in order to be conversant with their customers. In my experience, "regular" software developers often find that little of the subject matter strikes close to home for them, unlike financial or telecom systems, for instance. On the other hand, biologists and chemists with programming training often do not have enough experience in building large-scale systems.

### 3 Our Solution

The Informatics group at Affinium provides internal software development, bioinformatics and cheminformatics services. Unlike the biotech / pharma industry's tendency to outsource substantially [8], Affinium keeps a small contingent of full-time employees. The benefits have been lower costs and higher quality, greater user satisfaction with the group's products and availability, and overall better alignment with the company's goals. In line with industry practice [8], the group comprises 5 to 10 percent of the company's workforce.

Half of us have advanced biology degrees and programming training; the others have advanced computer science degrees, and learned basic molecular biology and chemistry on the job. The group's members have diverse software industry experience. Its manager (the author) is also the software architect; it is small enough for one person to manage this combination of roles. Overall, our makeup meets our needs.

Informatics always needs to address multiple projects, some of which have multiple customers. Affinium's ever-evolving drug discovery programs and collaborations require software of varying magnitude, which can be broken down into large-scale and small-scale development. This fuzzy distinction is made along the lines of duration, technology and maintenance. Small-scale development requires a single person for up to two months – for instance, writing scripts, software integration and small database development. Small-scale work is always done by Informatics members.

When large-scale software is needed, we compare building to buying-configuring-integrating. We take into account timelines, budgets, risk, vendor offering, and our skill-set and past experience. We have often found that scientific programs are better bought, whereas data and process management applications are better built, whether from scratch or using existing frameworks and third-party tools.

#### 3.1 Extreme Programming at Affinium

Our methodology has crystallized slowly since the group's formation two years ago. In the first few months it combined Waterfall and XP [1] ideas: frequent releases, automated testing, no over-providing, constant communication and feedback. Later, we gravitated toward more rigorous XP through learning from the industry and our own trial and error. The present adaptation, as it applies to mid- and large-scale development, is similar to the 19 practices described by Miller [7], combined with a new one, "architect". We also apply it to small-scale development, only with much less emphasis on automated testing, pairing and continuous integration. The next section outlines our methodology, broken down using the categories presented in [7].

##### 3.1.1 Joint Customer – Developer – Management Practices

As previously mentioned, we undertake a large number of projects for numerous customer groups. To be fair and agile in allocating our resources, we build our products iteratively. For new projects we use the Minimal Working Version technique: we plan iterations that culminate in a quick, good enough, usable 1.0 release, and dedicate part of our resources to its development. When the work is accepted, remaining stories and

maintenance work return to the project pool. The projects are rescheduled every two to four months to keep up with business changes. At any one time, we work on three projects at most. Ideally, each developer works on a single project during an iteration, completing other obligations between iterations.

The group works in an open environment with individual stations and low partitions, which we constantly talk over. It affords minimal privacy without hampering paired work or impromptu (“stand-up”) meetings of any number of people. Our customers do not sit with us: the scientists are in the lab and their managers are in cubicles or offices. Nevertheless, they are never more than a minute’s walk away.

The practice of retrospectives is relatively new to us. We have held only a few so far, most of them internal to our team. They helped us to air implicit disagreements and to express reasons for actions.

### 3.1.2 Developer Practices

We design and write our code with an automated-testing mindset, although sometimes we do not actually program all the tests. Typically, we code unit tests, but verify integration and acceptance manually. When deciding between automated and manual testing we weigh considerations of maintenance, schedule, difficulty, risk and available tools against ‘just enough’ manual testing. We determine the coverage and depth of either kind of testing by the modules’ operational profile, as well as schedule and risk. At the end of every release, we spend a couple of days on manual testing and bug-fix verifications. We delay releases when quality is at stake, e.g. data reliability, usability and severe bugs. We track all defects, fixed or not, in a central database.

The group follows an “architect” practice (not mentioned in [7]). While the developers take part in every product’s entire life-cycle, some tasks are centralized in the hands of the architect. Similarly to Fowler’s “Architectus Oryzus” [6], he is responsible for the vision and overall design of Informatics products, tracking design decisions, constraints and prospective development. He evaluates feature and change requests in light of existing and suggested designs. When competing alternatives are considered, he makes the final decision. He pairs with and mentors developers, reviews code, writes and enforces coding standards, and oversees high-impact refactoring (involving several modules). We find this centralization to balance collective ownership and the downsides of the “simplest thing that could possibly work” approach. For more on the thinking underlying this role, and how it scales, see [2].

We believe in collective ownership balanced by “disciplined action” [3]: one never introduces major changes without consulting first and notifying after. The reverse also holds: every necessary task will be undertaken by someone, like it or not. Anyone can improve or modify the code, but they must first consider code reuse and alternative designs with the architect, and check with the manager whether the schedule allows doing it then or later. Useful suggestions, designs and refactoring tasks, which are not undertaken immediately, are tracked for later consideration.

We work in pairs on all high-risk activities, such as product upgrades and live data manipulation. The developers often pair with the architect for requirements analysis, design and maintenance work, but only in some situations do they pair for coding and debugging. We feel that the architect and collective ownership practices, and our almost daily show-and-tell meetings, more than make up for lessened pairing.

We continuously refactor and integrate code, tests, data and documentation. We practice YAGNI (“You Aren’t Going to Need It”) by focusing on high-priority stories and doing just enough on them. However, being cognizant of the company’s needs, we can make some educated guesses regarding their future manifestation. Therefore, we always sketch initial design ideas for suggested features, refactoring and postponed fixes for major defects. These drafts are kept in mind when designing for a given release. Some unscheduled items never get done; some take long to be implemented, and may then be redesigned. All in all, our systems have undergone minimal course corrections.

### **3.1.3 Management Practices**

Given our multi-project, multi-customer situation, development projects are scheduled by a single person, the group manager. On one hand, he works with the customer groups on plans and priorities of maintenance and emerging projects; on the other, he plans iterations with the developers. The plans take into account the sustainable pace (“40-hour week”) practice, which the entire group feels strongly about.

Informatics members answer only to the manager, no matter what project they work on; the group has never been part of the company’s matrix structure. The manager is accountable for the group’s accomplishments and performance. He keeps its members informed through collective and individual feedback, including periodic performance reviews, and he presents goals and progress updates to senior management.

### **3.1.4 Customer Practices**

Software updates are rolled out typically every four to eight weeks. Each includes new and upgraded features, a planned set of bug fixes, needed refactoring and automated tests, and limited but sufficient manual testing. When a particular release targets a single customer group, they often plan it; in multiple-customer scenarios the development group plans the releases, as [9] advises. Despite their proximity, users’ exposure to new features may require a few weeks, so we normally plan two or three releases ahead. Later releases are planned shallowly, and adjusted in due time.

The input to release planning includes the big-feature customers’ decisions about story scope, the bug-fix and usability customers’ priorities, and our assessment of performance and stability. Our analysis of cost, risk and quality shapes the final plan (for instance, known defects may remain despite acceptance.) With less-involved customers, Informatics members with biology background serve as “customer proxies”.

We walk our customers through our designs, intermediate solutions and pre-release solutions. Similarly to the planning situation (and as per [9]), a single customer tests acceptance, whereas in multiple-customer situations, we do the testing.

## **4 How Our Methodology Has Worked For Us**

Overall, Informatics customers are pleased: the software we deliver fulfils their needs. It is deployed when still useful – keeping up with new drug discovery functions – and

important updates are rolled out rapidly. In this section I analyze successful aspects, problems and lessons we have learned from applying our methodology.

Many of the successes and lessons described below arose from our largest software undertaking, ProteoTrack™. It is a proteomics Lab Information Management System (LIMS), whose 1.0 alpha was developed by a consultancy using the Waterfall methodology. Two years ago the product was brought in-house, and Informatics has been applying the described methodology for its development and maintenance ever since.

#### 4.1 Successful Aspects

The approach of providing a Minimal Working Version using iterations, and reprioritizing further work, has been very successful whether for new applications, new major features, software integration or database development. Our customers have generally enjoyed receiving a usable product early, which included the functionality they needed. Although unused to small-functionality iterations, they quickly realized the benefits. For instance, early automation of data acquisition reduced manual labor, and properly devised data repositories made reporting simpler. Rapidly developed small increments indeed promoted timely feedback and enabled us to adjust to changing priorities ([1]). They also made for a gentler learning curve for our users, facilitating training and communication.

All group members, who have previously worked in small or large ‘mainstream’ environments, report higher satisfaction with our approach. Specifically, not having to follow slow, rigid processes, avoiding unnecessary work, and seeing all their work in use have been terrific motivators. They also report feeling more productive despite pairing and refactoring considerably, and working little overtime. Metrics we collected confirm that per developer, we have produced more features, fixes, Java classes and lines of code while being responsible for (“collectively owning”) a proportionally larger codebase. These metrics also indicate an almost constant pace.

It is worth noting that the group chose this Agile approach when the rest of the company, predominantly biologists and engineers, were set up as a matrix and following traditional goal-setting and project management practices. However, we are not at odds with anyone, as our accomplishments have justified our ‘unorthodox’ approach.

#### 4.2 Difficulties and Lessons Learned

One prominent difficulty we have had is defining the actual customer. Both Informatics and the scientists needed time to shift from the “programmers vs. users” approach to XP’s collaboration between providers and customers, and the shift is not complete yet. When 20 people will use a new feature, should the “customer” be just one of them? Selected four people? Their team leaders? We have tried to identify one or two representatives from each functional group, but this approach backfired several times due to disagreements or lack of participation. We have looked to leaders, people with a strong sense of product ownership, but very few felt confident enough to make decisions that affect other people. This insecurity was exacerbated when the company was formally organized in a matrix-management structure, which distanced people with

otherwise similar needs (and made holding retrospectives with them more difficult.) The Informatics group is thus often left to make decisions it is not qualified to make.

With heavyweight methodologies, developers are often committed to a single project for months on end. However with a short-iterations approach, the developers can bring several projects to a partial yet useful state. One problem is managing these multiple projects, each with its frequent releases, priorities and customers. Another is saying “no” to people when the load becomes too high. Lastly, a deployed partial project requires technical support, and occasional manual tweaking for yet-unsupported behavior. With more and more projects in this state, releases stretch longer and resources are spread thin.

We have tried to develop a common vocabulary with our user base, reinforcing it at meetings and training sessions. This approach has not been entirely successful, but the weakness of our common vocabulary has not impeded progress.

Several authors have written about retrofitting software projects to XP. Having to develop and maintain a Waterfall-style, no-tests alpha version, we have observed that non-trivial feature changes often merit a complete module rewrite rather than refactoring and retrofitting automated tests. We have applied both approaches to numerous modules, and the quality and productivity achieved by rewriting, especially by pairs, were often superior.

## 5 In-house development or paid consultants?

The benefits and disadvantages of engaging permanent employees vs. contractors are well known. This paper encourages the factoring of methodology into the equation, as more and more companies do these days. The many advantages of XP, as described above, hold whether the group is internal or external. However, our experience suggests that R&D environments, in which software needs vary and evolve rapidly, would do best to form in-house development groups who adopt or adapt XP. Significant gains include:

*Agility in resource allocation.* XP’s short cycles allow speedier redirection of resources when priorities change. Contractors would normally tackle a single project at a time, and charge a penalty if the project was stopped.

*Smoother process and data integration.* An organization often undertakes several software-related projects, which mutate over time to address changing needs and priorities. It can mitigate the impact of integration, maintenance and business evolution by centralizing architectural planning in the hands of a single, internal entity.

*Deeper understanding.* Most R&D environments are unique. Keen knowledge of an organization’s proprietary technology, various projects and processes is vital for focused, useful products. The customer can document and advise only so much; this knowledge is best found in internal groups who grow with the organization. Pair programming, collective ownership and architectural discipline help pass it on.

*More architectural options.* As XP encourages, we focused our efforts on the most commonly used features. We relied on manual intervention for infrequent activities, where development would give little return on investment. This approach would have been untenable had we paid someone else to write our software.

## 6 Conclusion

An innovative, scientific pursuit like structure-guided drug discovery has challenging needs for software whether the organization buys, outsources or builds it. Affinium uses a mixed buy-build strategy, the “build” part fully undertaken by the Informatics group, which resides outside the company’s matrix structure and bases its processes on XP. By frequently reprioritizing our project pool, we have been able to deliver timely, useful products and keep up with new discovery functions and needs. Incremental delivery has proven vital for early feedback, adjustment to changing needs, and motivation. Automated testing has been a blessing in the absence of testers. Just-enough design and refactoring have been critical for quick turn-around. It has not all been roses, however; we struggle with defining customers and their involvement, and it is difficult to manage multiple short-span, frequent-release projects.

I think the Affinium environment is reflective of other small companies who have scientific, research, development or production operations. Although our software must model the business – biology, chemistry and pharmacology in our case – it is no different to produce than other software. Its design and process must still consider usability, performance, developer and user interactions, maintenance, schedules and planning, and ultimately, value to the customer. I therefore suggest that XP can be customized to suit such environments, as our two-year-old case exemplifies. I propose further that an internal, organic team is in the company’s best interest, providing a more cohesive software experience than outsourced projects.

## Acknowledgments

I would like to thank all members of Affinium Informatics, past and present, for taking on this adventure together. We learn something new every day!

## References

1. Beck, K.: “eXtreme Programming explained”, Addison-Wesley, 2000
2. Broza, G.: Position paper for the “Experience Exchange” workshop at XP2003, online at <http://www.frankwestphal.de/xp2003/GilBroza.html>
3. Collins, J.: “Good to Great”, HarperCollins, 2001
4. Davies, K.: “The Demise of DoubleTwist”, online at [http://www.bio-itworld.com/archive/050702/survivor\\_sidebar\\_252.html](http://www.bio-itworld.com/archive/050702/survivor_sidebar_252.html)
5. DiMasi, J.A. and related Tufts research, online at <http://www.novartis.at/download/news/tufts/Tufts - PhRMA backgrounder.pdf>
6. Fowler, M.: “Who Needs an Architect?”, online at <http://www.martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf>
7. Miller, R.: “Demystifying Extreme Programming: “XP distilled” revisited”, online at <http://www-106.ibm.com/developerworks/java/library/j-xp0813>
8. U.S. Department of Commerce, “A Survey of the Use of Biotechnology in U.S. Industry”, online at [http://www.technology.gov/reports/Biotechnology/CD120a\\_0310.pdf](http://www.technology.gov/reports/Biotechnology/CD120a_0310.pdf)
9. Wallace, N., Bailey, P., Ashworth, N.: “Managing XP with Multiple or Remote Customers”, online at <http://www.agilealliance.org/articles/articles/Wallace-Bailey--ManagingXPwithMultipleorRemoteCustomers.pdf>