

# Do bigger telescopes need bigger software

Hilton Lewis<sup>a</sup>, Al Conrad<sup>a</sup>, and Bob Kibrick<sup>b</sup>

<sup>a</sup>W. M. Keck Observatory, Kamuela, HI 96743 USA

<sup>b</sup>UCO/Lick Observatory, University of California, Santa Cruz, CA 95064 USA

## ABSTRACT

We examine the factors that drive the size and cost of software for a hypothetical 30 meter telescope and compare those to that of a hypothetical 10 meter telescope. Our goal is to explore methods for developing estimates for such costs. The estimates provided here are intended primarily as examples for these methods, and should not be construed as accurate.

**Keywords:** extremely large telescope, software project cost estimation

## 1. INTRODUCTION

The designs for several extremely large optical/infrared telescopes (30-meters and larger) are currently under development. As software represents a significant component of such large telescopes, it is important to develop strategies for estimating the scope and cost for that software.

Our method for estimating the costs of the software for a 30 meter telescope is shown in Fig. 1. We begin with an estimate for the software cost for a 10 meter telescope and then apply adjustments for several factors.

## 2. ESTIMATE FOR A 10 METER TELESCOPE

We apply the experience of the Keck 10 meter telescope as a benchmark for estimating the cost of a hypothetical 10 meter. We partition the software into six components, as illustrated in Figure 2.

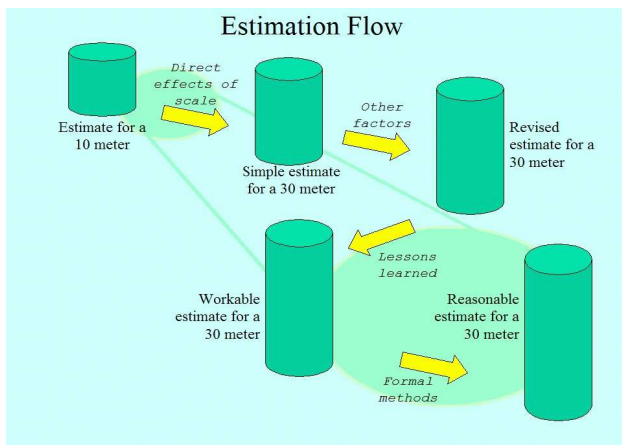


Figure 1. Estimation method

Component	Person Years	Lines of Code (x1000)	Lines / Day
Telescope Controls	<b>20</b>	<b>600</b>	<b>115</b>
Instrumentation (5)	<b>60</b>	<b>1000</b>	<b>64</b>
Active Optics	<b>16</b>	<b>110</b>	<b>26</b>
Adaptive Optics	<b>18</b>	<b>200</b>	<b>42</b>
Infrastructure	<b>10</b>	<b>190</b>	<b>74</b>
Coordination & Management	<b>10</b>	<b>N/A</b>	<b>N/A</b>
Total	<b>134</b>	<b>2100</b>	<b>60</b>

Figure 2. Breakdown by subsystem

We then applied one method to estimate the amount of labor required to generate the telescope software, and another to determine the total lines of code. Labor estimates (in person years) were derived using the corporate memory, i.e., the recollections of the various software managers who oversaw the development of the major subsystems. Lines of code were determined using a simple tool (the Unix program *wc*) to count the

Further author information- (Send correspondence to H.L.)

H.L.: Email: hlewis@keck.hawaii.edu Telephone: 808-885-7887; Fax: 808-885-4464

number of source lines in the relevant source directories. Comment and non-comment lines were counted the same, and no distinction was made between human-written code versus automatically generated lines of code, such as that produced by the EPICS *capfast* program. The results of these estimates for each subsystem are shown in Fig. 2.

We next explore each of the various factors that are applied in sequence to map the estimate for a 10 meter telescope into one for a 30 meter telescope.

### 3. DIRECT EFFECTS OF SCALE

The direct effects of scale fall into three categories: software cost increases that are directly related to larger optics and mechanical structures, software cost increases that are directly related to a new class of instruments, and software cost increases that are directly related to the rapidly developing field of wave front correction.

#### 3.1. Larger optics and mechanical structures

The most obvious effect of larger optics and mechanical structures on software costs stems from an increase in the number of mirrors which will be comprised of actively controlled segments, and an increase in the number of total segments comprising the primary mirror. The Keck telescope, used as the baseline for the cost estimate given in section 2, uses an unsegmented secondary and tertiary, and a primary consisting of 36 hexagonal segments. By contrast, the current concept for the 30 meter CELT telescope<sup>1,2</sup> proposes segmentation for the primary, secondary, and tertiary, with 1080 segments in its primary.

Moore's Law<sup>3</sup> suggests that the basic architecture used to perform the calculation to control 36 segments using 1990 processor speeds can again be used to control 1080 segments using 2005 processor speeds\*. The 30-fold increase will however require more actuators, sensors, and communication hardware. With this increase in scale, fault tolerance via redundancy, and the software to handle fail-over, is implied. The visualization tools used to operate and monitor the health of the system will require new designs that accommodate the large increase in actuators, sensors, and communication hardware.

The scale-up in mechanical structures will bring an increased reliance on active flexure control systems. This is an effect seen at Keck and other large telescopes, particularly within new, larger instruments. Active flexure control systems typically contain a large software component.

#### 3.2. New class of instruments

A higher percentage of instruments for a telescope like CELT will be behind AO, and, of those AO instruments, there will be a higher percentage with integral field units.<sup>4</sup> AO instruments require more complex interaction with the facility. An IFU instrument requires a sophisticated data reduction pipeline. Each of these effects translates into higher software costs for thirty meter instrumentation than the estimate for ten meter instrumentation given in the previous section.

More sophisticated detector technology will appear on a future thirty meter telescope. For example, superconducting tunnel junction devices appear poised to make their way into ground based astronomy in the coming decade. Such devices will require new read out software which measures both the location and energy of incoming photons.

Faint object spectroscopy will continue to be a driving force behind large ground based telescopes and there will be more and bigger slit masks available on the instruments. Slit mask alignment software will require more sophistication to accommodate the increase in the number (as much as 1000)<sup>4</sup> slits. Fiber positioners will be more common. Each of these implies increased software costs.

---

\*It will not be necessary to use special purpose (e.g., DSP) compute engines like those required for today's wave front controllers.

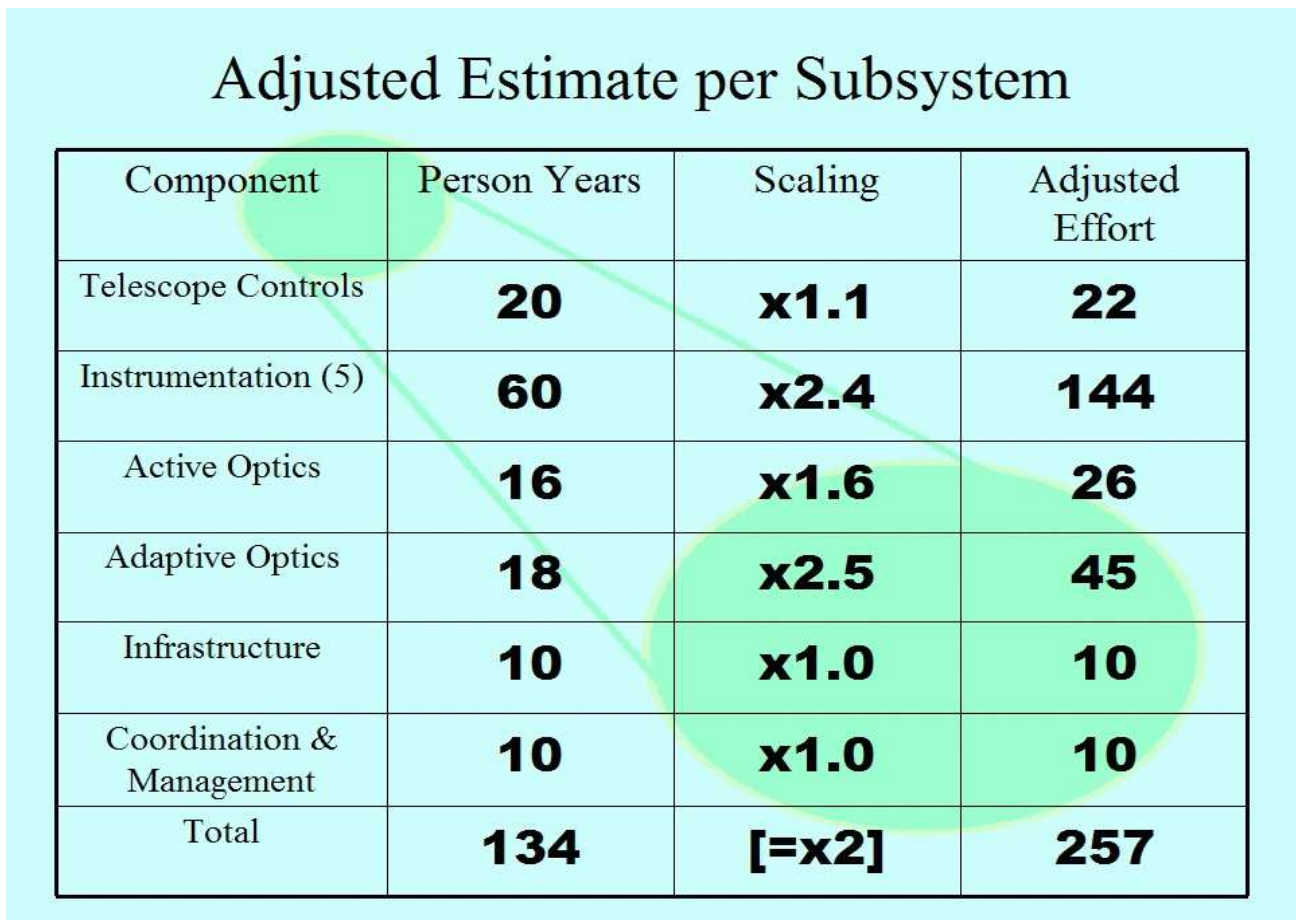
### 3.3. Wave front correction

Many features will exist in the adaptive optics of a future thirty meter telescope which were not accounted for in the baseline given in section 2. Multiple lasers will be used and these will be next generation lasers which require new control systems. The effects of multi-conjugate AO and extreme AO on the software costs for a future thirty meter will also be significant.

Finally, in addition to adaptive optics, “seeing limited” instruments will require telescope wave front sensing. Although this capability is common on the more recent 8-meter class telescopes, it was not developed for Keck and was therefore not included in the cost estimate baseline given in the previous section.

### 3.4. Adjusted Estimate per Subsystem

We looked at factors like those described above and estimated the increase in total effort, per subsystem, as compared to the person years measured for the baseline. The results of these estimates, which reflect a two fold overall increase, are provided in Figure 3.



The table is titled "Adjusted Estimate per Subsystem" and is set against a light blue background. It contains four columns: "Component", "Person Years", "Scaling", and "Adjusted Effort". The data is as follows:

Component	Person Years	Scaling	Adjusted Effort
Telescope Controls	<b>20</b>	<b>x1.1</b>	<b>22</b>
Instrumentation (5)	<b>60</b>	<b>x2.4</b>	<b>144</b>
Active Optics	<b>16</b>	<b>x1.6</b>	<b>26</b>
Adaptive Optics	<b>18</b>	<b>x2.5</b>	<b>45</b>
Infrastructure	<b>10</b>	<b>x1.0</b>	<b>10</b>
Coordination & Management	<b>10</b>	<b>x1.0</b>	<b>10</b>
Total	<b>134</b>	<b>[=x2]</b>	<b>257</b>

Green circles highlight the "Person Years" and "Adjusted Effort" columns. Green arrows point from the "Person Years" column to the "Adjusted Effort" column for each row, illustrating the calculation of the adjusted effort.

Figure 3. Adjusted estimate per subsystem.

## 4. OTHER FACTORS

This section serves as a catch-all for those factors affecting software costs that do not fall into the broader categories covered elsewhere. These are: safety issues, multiple stake holders, underlying engineering, data management, and contingency.

#### **4.1. Safety Issues**

As the cost of the facility increases, so the investment in more sophisticated software to address underlying safety issues becomes more important. Traditionally at observatories, system safety has largely been the realm of hardware, but this is already impractical for some of the more complex sub-systems being deployed, where the underlying complexity has increased the number of possible failure modes many times. Other factors driving this increased investment are the higher potential cost of accidents involving bigger, more expensive systems. Just as important is the increased value placed on the availability of the facility. In addition, the high profile of a facility costing an appreciable fraction of a billion dollars renders down-time even more unpalatable.

#### **4.2. Multiple Stake-holders**

A large facility will of necessity involve multiple stake-holders, who require more insight into and involvement with the development process. The additional reporting, reviews and consultation needs increase the management cost of the program across the board. A clear division of responsibilities and strong central management is needed to ensure that the effort needed to fulfill these requirements does not become excessive.

#### **4.3. Underlying Engineering**

The quality of engineering in the other disciplines affects software cost. For example, a disciplined application of finite element analysis during the mechanical design phase may preclude the need for applying a software flexure control system after the fact. The “fix it in software” syndrome can have a significant impact on software costs since these fixes are often poorly understood and come late in the game, when the cost of software changes is much higher.

#### **4.4. Data Management**

The sophistication and complexity of instruments capable of taking full advantage of a large telescope demands a comprehensive data management suite. Such a suite of tools includes pre-observation tools such as those for proposal preparation, planning tools and even possibly weather forecasting as well as post-observation tools such as data quality monitoring, archiving and data reduction pipelines.

Archiving tools (for storage, cataloging, retrieval and distribution) are central to data quality monitoring as well as for safety backups, engineering support and long-term science support.

Data reduction pipelines are needed for individual instruments, as well as for facility-wide capabilities such as adaptive optics, for such areas as estimating AO point spread functions and high-contrast PSF calibrations

#### **4.5. Contingency**

The proper estimation of contingency is very difficult, especially given the nature of this kind of enterprise, involving several novel technologies. Nevertheless, a systematic approach to computing contingency can be taken, where contingency is computed “bottoms-up” for individual elements, based on a risk assessment of the individual elements. Just as important as computing contingency though is the need for active risk management of the program, particularly the more speculative parts. Careful attention has to be paid to real requirements, to ensure that costs are not driven excessively by goals that, while interesting and valuable in their own right, are not crucial to the success of the overall program.

#### **4.6. Overall Effect of Other Factors**

The overall affect of the factors listed above were estimated to contribute an additional 50% to the software cost. Of all the flows in the path, this one is the most uncertain. The need for contingency and the possible effects of the “fix it in software” syndrome are particularly unpredictable.

### **5. LESSONS LEARNED**

This section describes some of the lessons learned in the development and maintenance of the Keck Observatory software in particular, but is also based on observation of the development practices at other large telescopes.

## 5.1. Staffing Levels

Software development teams for ground-based astronomy in the USA have in the past usually started out far too small. This has led in particular to problems in developing and adequately supporting the infrastructure needed for distributed development teams. Other problems that surface are inadequate testing, premature obsolescence and inadequate readiness for operations. The net result is a drain on the staff who are pressed into “fire-fighting” to support prematurely deployed software systems.

Delays in development in turn can delay the entire program, leading to (lost) opportunity cost as other telescopes and instrumentation come on line, exploiting the scientific niche for which the system was designed.

There are however clear signs that the community as a whole is starting to recognize and deal with these problems. We address some of these key areas in the sections that follow.

## 5.2. Investment in infra-structure

A rich and reliable infrastructure is essential to support a geographically distributed project, with diverse developers of differing skill levels. A good infrastructure can drive the cost to market (the deployment cost) down, although this is sometimes counteracted by increasing the scope of individual applications to take advantage of the available facilities. Unquestionably though, when a common philosophy is used across the board, the life-cycle cost is reduced and maintenance of the entire system is simplified.

The core infrastructure must be developed and deployed early. Too often, such infrastructure, while ambitious in scope is poorly executed, leading to frustration and delays on the part of the application developers and, in extreme cases, to the abandonment of parts of the common infrastructure.

The best approach is to provide custom-developed components and to specify off-the-shelf components (such as CORBA ORBs). This infrastructure must meet real needs, not just satisfy some aesthetic ideals of the design staff. Such needs of course extend beyond satisfying immediate functional requirements; an example of this is long-term maintainability. Care must be taken to conduct research and development during the conceptual design phase, to limit schedule and budget uncertainty (and hence risk) during the later design phases. In these later phases, the approach should be to adopt reasonably stable and evolved components wherever possible.

Finally, enough effort must be allocated to properly support the infrastructure once it is deployed, to maintain and evolve the core components, to manage upgrades and most importantly, to provide support to application developers - ranging from simple “how to” inquiries to more complex questions regarding customization of the supplied components. In several telescope projects, this aspect of the infrastructure has been under-estimated, leading to a failure to adopt the infrastructure across the board. This in turn has led to delayed implementations with ad-hoc solutions to common problems.

## 5.3. Interface control documents

The desirability for clear, stable, configuration-controlled interface control documents (ICD) is obvious for the development of software in a distributed environment. However, a less obvious but equally important rationale for ICDs is that they form the underpinning for any sub-system testing. Any violations of an ICD therefore compromise the validity of the testing conducted, leading to re-testing at a later stage when changes are more expensive.

## 5.4. Simulators and subsystem testing

It is well known that good simulators (hardware and software) are essential to adequate testing of software. It is equally well known particularly in telescope development circles, that they are frequently among the first components to be dropped when schedules and budgets get tight. It should be borne in mind that simulators are not only needed in the development phases, but also in the later software maintenance phases, which frequently form the bulk of the life-cycle cost of a delivered software system. In these later phases, access to the telescope becomes difficult, not only because there are many competing activities (not least of which is observing!) but also because of an increasing reluctance to change the configuration of a stable system (with the associated risk of something going wrong) in order to test new software.

The ability to independently test subsystems is essential, even though this requirement may drive the overall design. This is particularly important in the early commissioning phases, but will continue to be required later, when the full facility may rarely be available.

## 6. MODERN SOFTWARE PRACTICES

Modern software engineering practices<sup>5</sup> are effective at reducing risk in terms of schedule and performance, rather than reducing the cost of development per se. Frequently these practices come with an apparent larger price tag up front; this is largely because they make it harder to indulge in wishful thinking about the true costs of developing software, in particular the costs associated with developmental risk (computed as the product of schedule uncertainty and component cost). However, just as in the case for infrastructure cited earlier, the overall life-cycle costs are reduced through higher reliability and easier maintenance of the system.

### 6.1. Design

Of the many design principles that apply, we will mention only two: design patterns and interaction design.

Design patterns<sup>6</sup> describe solutions that have evolved over time. There are many documented design patterns for standard software problems. Although design patterns exist in the astronomical software domain (for example in computing telescope pointing), these are communicated by word of mouth and coding examples, rather than in a standardized style. There is much to be gained in codifying these patterns, not only for the benefit of future developers of telescope software, but also for the future maintainers of the system.

Interaction design<sup>7</sup> concentrates on the interaction of users with the underlying system, and focuses on empowering users to accomplish their work in a natural and intuitive fashion. Interaction design should be viewed as a core design activity, potentially affecting key architectural choices. It should be led by staff not involved in developing the software, but by those who have a deep understanding of the problem domain. This activity is distinct from user interface design, which is concerned with issues such as human factors, look-and-feel, presentation, and the separation of form and function (the model-view-controller design pattern).

### 6.2. Development process

Prototyping poorly understood areas is a well-known and useful technique to mitigate risk, while at the same time clarifying requirements for the system.

The development process itself should be iterative and incremental, starting with the implementation of the core requirements, with successive iterations being fully integrated with previous ones and delivering successively greater functionality. This approach allows for more successful management of the process, allowing early identification of problems and the ability to change scope or add resources to deal with issues earlier in the development cycle, when changes are simpler and costs are lower.

Reviews should demonstrate critical technologies and processes early. This provides essential feedback on performance and usability early in the process, as well as providing management with key information on the team's capability. This approach does not supplant the more traditional document based reviews; rather it complements them, providing useful feedback to the team and the program management.

### 6.3. Testing

Formalized testing is a well-understood and expensive process. We suggest that these costs can be reduced by capitalizing on the unique aspects of development of astronomical software, namely that the scientists and engineers can and do play an essential role in specifying, reviewing and testing software and that the development team contains members that have an intimate understanding of the problem domain and are personally committed to the success of the larger program.

We advocate that testing be explicitly costed and managed as part of the development activities. Furthermore, test plans and software need to be developed concurrently with other development activities. The software should be tested by an independent test team, using formal test methodologies (such as regression testing, black-box and white-box testing) and with full traceability to the software requirements.

#### 6.4. Estimate of the cost of adopting formal methods

We estimate (very crudely, and by analogy with our experience in developing software for other astronomical applications) that the result of applying formal methods would increase cost by order 10% of the software cost. The increased cost will chiefly occur in the areas of design and formal testing.

Once we adjust for the costs associated with modern software practices, we arrive at the end product of this estimation method, which yields the estimated cost for the software for a 30 meter telescope to be on the order of fifty million dollars (see Fig. 4).

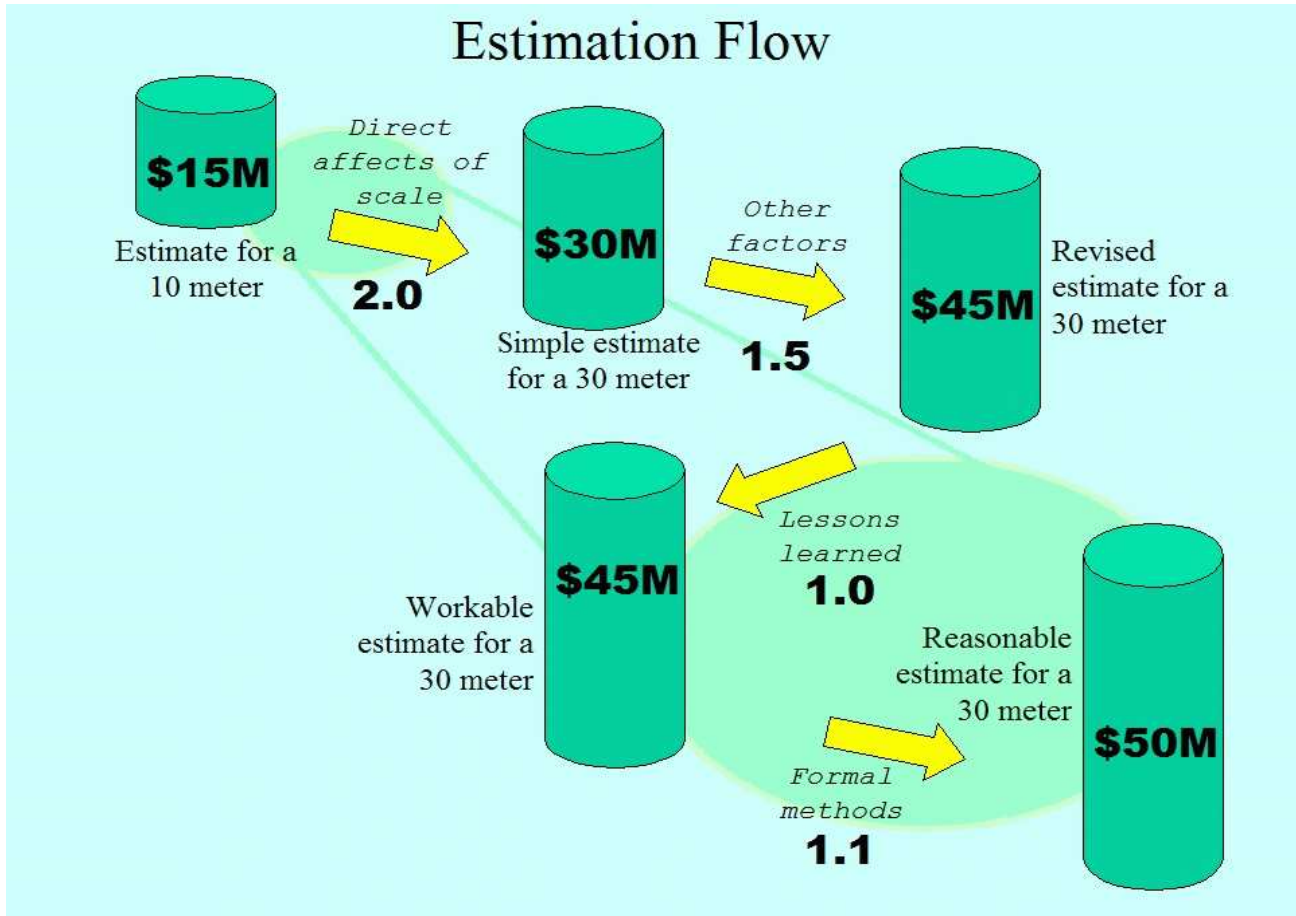


Figure 4. Estimation method applied

### 7. CONCLUSION

We start by indulging in some crystal ball gazing.

We expect that formal risk analysis will become an integral part of the software management toolkit for the development of astronomical software, particularly for large and complex systems. This is no surprise - it is already standard in many industrial and commercial applications.

We expect that even the core team will be distributed geographically. This is becoming a necessity in order to obtain the right mix of skills, but is also becoming increasingly practical with the universal advent of high-bandwidth connectivity. Such a team will need to make extensive use of sophisticated tools and "virtual teams" to achieve the synergy needed to be effective.<sup>8</sup>

A much greater understanding of software issues and cost drivers will be present at the executive management level in astronomy. This is largely a reflection of the increasing importance of software as a cost and quality driver in ground-based telescope enterprises. The result will be budgets and schedules more attuned to the realities of software development, and less of a “glass and steel” approach to management.

Finally we expect that the correct application of new software technologies and practices will not of themselves reduce delivery costs, but will go a long way to making the software development process more predictable, while at the same time greatly improving overall software quality: in particular performance, reliability, and maintainability.

We conclude that indeed, larger telescopes will require more sophisticated, complex and expensive software. However the benefit is that it maximizes the return on the capital investment, allowing astronomers and engineers to take full advantage of the unique opportunities afforded by the next generation of ultra-large telescopes.

## ACKNOWLEDGMENTS

We wish to acknowledge the contributions of Jerry Nelson, Terry Mast, and Richard DeKany in interactions in the preparation of the CELT conceptual design report (CELT report #34, also known as the “The Green Book”).

The W. M. Keck Observatory is operated as a scientific partnership among the California Institute of Technology, the University of California and the National Aeronautics and Space Administration. The Observatory was made possible by the generous financial support of the W.M. Keck Foundation.

## REFERENCES

1. *California Extremely Large Telescope Report 34: Conceptual Design for a Thirty-Meter Telescope*, University of California and California Institute of Technology, 2002.
2. J. E. Nelson, “CELT Rocks,” in *Future Giant Telescopes*, J. R. P. Angel and R. Gilmozzi, eds., *Proc. SPIE* **4840**, 2002.
3. B. Schaller, “The Origin, Nature, and Implications of Moore’s Law,” <http://mason.gmu.edu/~rschalle/moorelaw.html>, 1996.
4. K. Taylor, “Instrumentation for the CELT 30 meter Telescope Project,” in *Instrument Design and Performance for Optical/Infrared Ground-Based Telescope*, M. Iye, ed., *Proc. SPIE* **4841**, 2002.
5. I. Sommerville, *Software Engineering*, Addison-Wesley, Menlo Park, 2001.
6. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Menlo Park, 1995.
7. A. Cooper, *The Inmates are Running the Asylum: Why High-Tech products drive us crazy and how to restore the sanity*, SAMS/Macmillan Computer Publishing, Indianapolis, 1999.
8. A. Conrad *et al.*, “Applying modern collaboration methods to distributed engineering projects,” in *Advanced Global Communications Technologies for Astronomy II*, R. Kibrick, ed., *Proc. SPIE* **4845**, 2002.