Dynabook Redux

Dr. David West – Dr. Margaret Young – Dr. Jane Quillien
Department of Business, Media, and Technology
New Mexico Highlands University
Las Vegas, New Mexico, USA
dmwest@nmhu.edu -- young_m@nmhu.edu

Abstract—**We describe an innovative new program in software-driven systems design. The program is experience driven and competency based. Students are apprentices in the traditional sense of the term. The goal is to establish and nurture a community of master software developers / system designers. This program creates some unique support needs as well. Core to meeting these needs is a tablet based computer with curriculum delivery, collaborative development support, communication support, and intra-community social networking tools. We describe the development of the desired support tool and link those efforts to the Alan Kay (1968) depiction of a Dynabook.**

Education, dynabook, community, web learning, software

I. INTRODUCTION

In 1995, New Mexico Highlands University (NMHU) piloted a new program in software development. The program focused on experiential learning and demonstrated mastery of approximately 500 competencies instead of courses. The pilot ran for one year with some remarkable achievements: 100% freshman retention, 50% female and 85% minority participation, 100% placement rate for all eligible students at pilot termination, and 50% of students (14) published papers at premier refereed conferences.

Today, NMHU has evolved the pilot into both a Bachelor and Master degree program in Software-driven Systems Design (SSD). These programs are radical departures, both in focus and structure, from existing computer science, informatics, software engineering, and management information systems degree programs.

Both programs present significant challenges. Textbooks, for example, are of little use because the program subdivides the curriculum into smaller units than the typical semester or quarter course, and the delivery of those units is "on-demand." In the absence of a support tool capable of enabling the learning, development, and social networking objectives of the program it was necessary to design and begin development of the needed device. The inspiration was two-fold: the Dynabook [1] concept of Alan Kay and the fictional "young ladies primer" described in Neal Stephenson's book, *Diamond Age*[2].

Support for the completion and presentation of this paper was supplied by Los Alamos National Laboratory.

II. SOFTWARE-DRIVEN SYSTEMS DESIGN

It is useful to begin with a discussion of the degree programs with an emphasis on their goals and the ways in which they are different from the typical higher education program.

A. *Program / Degree Focus*

Computer Science (CS) and Software Engineering (SE) are all about the 'artifact' – the computer and the program. The Software-driven Systems Design (SSD) is all about the system in which computing artifacts (hardware plus software) are deployed and the effects on the system that arise from that deployment.

"Systems," in CS and SE are mechanical, deterministic, and complicated – an idea borrowed from classical (pre-quantum) physics. Systems in SSD are living, highly dynamic, adaptive, and complex.

For CS and SE, problems to be solved are subject to formal definition and precise requirements and are solved by constructing a machine that satisfies requirements. For SSD the problems to be solved are ill-formed, often vaguely defined, lacking sufficient information, and often subject to redefinition as a function of increasing knowledge of the problem and potential solutions. This kind of problem solution requires thinking skills beyond "computational thinking" – skills that include metaphoric reasoning and design thinking.

SSD differs from Management Information Systems (MIS) as well. Both programs are concerned with 'process' and 'management' issues applied to the act of software creation and deployment. MIS is grounded in the philosophy of software engineering and the prevailing business philosophy of 'scientific management." SSD is grounded in ideas of 'art,' 'craft' and 'coaching.'

SSD values "Thriving on Chaos [3]" and "Embracing Change [4]" instead of "Planning the Work and Working the Plan."

B. *Curriculum*

SSD, CS, SE, and MIS share some learning goals. In all four areas it is necessary to understand the nature of a program, programming languages, data constructs, modularity, and well-

known formal or patterned solutions like algorithms and mathematical applications.

Unlike CS or SE, SSD does not emphasize the theory of computation, proof, or machine level software (including operating systems, compilers, device drivers, or network infrastructure).

SSD adds curricular material including topics like best practices, tools, communications (written, oral, visual), reading (both purposeful and for pleasure, following the dictum of Alan Kay, "Those that do not read for pleasure cannot read for purpose."), and the concepts and history behind those topics. Significant amounts of material from other disciplines including: cultural anthropology, philosophy, history, sociology, psychology, music, design, and mathematics; are part of the SSD curriculum.

The breadth of the curriculum is more than would normally be encountered in two majors or two graduate degrees. It is possible to deliver this volume of content because of the way it is modularized and delivered. Curricular modules are roughly equivalent to 16-32 hours of student-material interaction. Instead of 16-20 three credit hour courses, students are responsible for mastering 150+ discrete learning modules.

Each module is self-contained and Web-based. Each module incorporates background material, integrative links to other modules, topical content, exercises and examples, usually video 'lectures,' and self-evaluations. A 'final exam' completes the module. Completion of modules is self-paced.

Each topic area (each module) roughly equates to a competency. Each competency is assessed at seven different levels. Completion of the on-line module establishes competency level 1 – "concepts and vocabulary." The other six competency levels are:

- 2- demonstrate application of knowledge under supervision;
- 3- demonstrate application of knowledge independently;
- 4- demonstrate application of knowledge in a different domain or problem context;
- 5- mentor others in the application of the knowledge;
- 6- improve or create instructional materials;
- 7- make an original contribution (whitepaper, conference paper/presentation, or publication) to the topic area;

These competencies are assessed in the context of a Studio experience - while working on real world projects on behalf of paying customers.

Each learning module is delivered "on demand" in the context of a project. Students are motivated to learn the material because they need it to advance the project (for which they are paid as apprentices). They are able to immediately apply the knowledge – ensuring retention – and to integrate it with other elements of knowledge thereby promoting integration of knowledge from diverse subject areas. The effectiveness of this approach can be anecdotally illustrated by a student in the pilot program, a freshman, that entered the program with zero computer knowledge or experience (could not cut and past in a word processor) and a semester later was mentoring other students in Java and J2EE on a project for the New Mexico State Engineer's Office.

Each module is structured on a modified pattern format [5] [6]:

- Context – in which the knowledge is found and found to be useful
- Problem – the kind of issues or problems that are amenable to application of the knowledge
- Knowledge description – the substance of the module
- Examples – how the knowledge has been found useful plus examples of applications (e.g. in program code) of the use of the knowledge
- Variations and Connections – themes, extensions, elaborations, and contrarian positions with regard the knowledge, plus connections to other learning modules and areas of study.
- Resources – Web, book and paper references.
- Self-evaluations.
- Final Exam

III. THE STUDIO

The core of the SSD program is the Studio. Students (and faculty) are expected to spend roughly thirty-six hours a week in the Studio. The Studio is itself a "one-room schoolhouse" with everyone from freshmen to graduate students in one room at the same time.

The primary activity in the Studio (roughly 65% of the time) is focused on development projects. Students use an Agile approach (exploratory, iterative, incremental) and work side-by-side with professional developers (thereby gaining the kind of tacit knowledge of the discipline that is normally learned only after leaving school).

Other activities in the Studio include student-instructor interaction and feedback on the learning modules, individual and small group "learning spikes" (the closest approximation to typical lecture/discussion), and weekly reading and writing workshops [7]. All work in the Studio – development and learning – is done is pairs and small teams (another Agile derived practice). All work is collaborative and students learn a teamwork model instead of the typical individual competition model of most higher education.

Students move through a variety of different roles in the Studio, including: developer, tools maven, mentor, coach (we do not have project leads or project managers per se), designer, customer liaison, tester, and systems administrator. Students accumulate a portfolio of commercial grade work that they have completed or to which they have made significant

contributions. They also have established ties with the professionals working in the Studio and with companies that sponsor projects. Graduates have a transcript, a portfolio, a professional network, and a reputation. This makes them highly employable when they leave the program.

IV. COMMUNITY

An implicit objective of this program is to initiate the formation of a "community of practice." In the most trivial sense, this means a professional social network akin to Linked-In. But it differs in significant ways. First, it is a closed community – only those who have participated in the program – as student, mentor, project sponsor, or faculty. Second, it provides participants with a "reputation" based on empirical shared experience – providing a very real kind of "certification."

The software world has traditionally been subject to fads – in technology and in methodology. As soon as a fad takes hold, seemingly everyone immediately claims to be an expert in that 'new thing.' This is of course, not true, and some means of sifting the wheat from the chaff is required. The traditional answer has been certification. But certification suffers from the fatal flaw of being based solely on one's ability to pass a test – occasionally with the addition of some kind of work sample.

In the SSD community you will have a reputation – a personal kind of certification. Some number, 'X,' of the 'Y' number of people who have worked at your side, under your direction, or as your direct supervisor or professor personally attest to the fact that you are a rank 'Z' programmer/analyst/network engineer/or other title. This reputation indicator augments your profile – your level rating on each of the several hundred competency factors addressed by the program.

It is necessary for the SSD community to expand rapidly – as the value of the education becomes widely known, the demand for members of the community will increase rapidly. These are the same forces that give rise to the prevalence of poseurs – self-proclaimed experts in any new technology or methodology. The kind of rapid expansion necessary comes from the ability to establish a large number of Studios around the globe. Students will be exposed to a common curriculum, common Studio practices, and will be required to physically participate in work at Studios other than their local Studio. (Approximately 1/3 of the time spent in the program will be at a Studio other than the one in which the student initially enrolls.) Studios will also be connected and project teams will frequently (probably most of the time) involve participants from multiple Studios across multiple time zones.

V. TECHNOLOGY SUPPORT OR DYNABOOK REDUX

The vision, strategic goals, and tactical objectives of the SSD program requires substantial technical support. Bits and pieces of the required support currently exist: e.g. laptop computers, remote desktop software, and interactive television (ITV).

While it is possible to amalgamate multiple technologies to service the combined needs of the SSD program, it is not necessarily desirable to do so. Issues of platform (chip set, OS, browser, etc.) compatibility and interoperability remain serious obstacles. The cost of licensing the plethora of technologies needed to support the vision, goals, and objectives, in their entirety, is daunting.

What seems to be needed, and what we are actively engaged in creating, is an "SSD Tablet." The inspiration of this device is Alan Kay's Dynabook.

As Kay described it, in 1968 – well before any of the technology necessary to implement it was available or even imagined – the Dynabook would be an interactive computing device to support the education of children. The form factor for the device was roughly that of a typical hardcover book.

Interactive lessons would be delivered via this book along with the ability to access any information that the child might need to enhance their learning and slake their innate curiosity. The device would be "programmable" in the sense that the child could "dialog" with the computer in order to change its nature or extend its capabilities. Several years later these same objectives shaped the Smalltalk programming language (and current variants like Scratch).

The Star project at Xerox Palo Alto Research Center (PARC) embodied many of the ideas of a Dynabook, except for form factor (it was a desktop) and conversations between child and computer were still mediated by keyboard and mouse.

Today we are much closer to realizing the Dynabook concept. When Steve Jobs showed Alan Kay the first iPhone, Kay said that it was the first approximation of the Dynabook worthy of criticism. (One criticism, the form factor was too small. Kay suggested a form factor of a Moleskin notebook he was carrying. A year or two later, the iPad.)

The ubiquity of wireless networking and the availability of virtual resources (Cloud Computing) via those networks are also critical infrastructure technologies that make it possible to realize the Dynabook concept.

VI. SSD TABLET

We have decided to build a special purpose tablet computer-based application that will support the SSD Program: a tablet for mobility and for reasonable form factor; special purpose to optimize the degree of support for the program. Some initial expectations for the device include:

- Security and access control based on encrypted communications and URL, embedded hardware id codes, plus user biometrics – all unobtrusively and periodically verified while the device is in use.
- Simplified intra-community asynchronous communication and resource sharing (email and document sharing analogs)
- Simplified intra-community synchronous communication (e.g., chat, video conferencing, VOIP.)

- Augmented collaborative work tools (e.g. shared whiteboards, remote desktop, distributed modeling tools).
- Delivery of multimedia educational content. Content would be designed for form factor compatibility.
- Support for software development including an IDE and Bluetooth keyboard and mouse as well as touch screen and voice recognition.
- Portfolio (student accomplishments) access.
- Assessment and reputation sharing.
- Contact lists, calendars, and other productivity tools.

This basic functionality of the SSD Tablet is extended by making it a component in a Studio ecology. Other components would include multiple projectors (standard and pico) and multiple cameras mediated by devices like AppleTV, WiFi and Bluetooth. Each Studio becomes an Ambient computing environment with control information coming from sensors built into devices (e.g. accelerometer and gyro compass in an iPhone) or camera captured gestures from participants in the room and information captured from and displayed on any surface (with intermediation by appropriate software as desired). A simple gesture on a touch screen would cause the contents of the tablet display to be projected on a wall and a wave of the arm could send that information from one all to another (and potentially to a wall in a Studio on the other side of the world).

A final aspect of the SSD Tablet is inspired by Neal Stephenson's description of the "young lady's primer," in his book *Diamond Age*. The primer was a Dynabook on nanotechnology steroids. It served all of the purposes of a Dynabook coupled with a sophisticated (far beyond current capabilities) AI and equally sophisticated voice recognition/generation tools (so it could read the young lady a bedtime story), and connection to a network of behind the scene human beings. The individuals in this network worked from their homes or wherever they happened to be and could monitor a specific primer, its environment, and its user. Because they were human, they could recognize situations of potential danger and cause the primer to initiate actions or alert the young lady to take actions to ameliorate the situation. The human "listener" could also detect moods, puzzlements, and other aspects of the state of mind of the young lady and could initiate conversations or otherwise interact with the young lady in an appropriate manner. Someone in the network was available to any young lady – via her primer – twenty-four / seven.

Imagine how the community goals of the SSD program would be enhanced if a similar kind of person-to-person connectivity could be implemented in the SSD Tablet.

Another aspect of the tablet merits brief mention. If the membership goals of the community are realized (thousands of members and hundreds of Studios) and as the curricular content grows to anticipated levels, there will be a need to supplement access with the use of AI-type tools. Most notably some kind of "intelligent search" capability, some basic machine learning capabilities to facilitate workflow, and features akin to code completion in program editors, only focused on design models instead of code. The last feature noted would also assure effective use of design libraries and design patterns. We are aware of this AI type potential but have yet to actively explore how it might by implemented and utilized.

VII. CONCLUSION

Initially, the SSD Tablet will be a mash-up of independent applications executing under the auspices of a browser or OS defined platform. The longer term goal is to get even closer to the Dynabook concept by providing a single environment that serves platform, application, development and interaction requirements, e.g.: a small firmware virtual machine interacting directly with the hardware (no intervening operating system), a single modularization model/metaphor (the object), a single programming language, and a library of object components that individually and collectively provided all application and IDE functionality. The most obvious candidate for enabling this goal is Smalltalk (actually one of the dialects of the one of the current incarnations of Smalltalk – Squeak) – which reflects many of the same ideas and values expressed for the Dynabook.

REFERENCES

[1] http://www.mprove.de/diplom/gui/Kay72a.pdf

[2] JN. Stephenson, Diamond Age: A Young Lady's Illustrated Primer, NY: Spectra, 2000.

[3] T. Peters, Thriving on Chaos, NY: HarperCollins, 1991.

[4] K Beckand C. Andres, Extreme Programming Explained: Embrace Change, second edition, NY: Addison Wesley, 2004.

[5] http://martinfowler.com/articles/writingPatterns.html.

[6] http://en.wikipedia.org/wiki/Software_design_pattern#Structure.

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[8] R. Gabriel, Writer's Workshops & the Work of Making Things: Patterns, Poetry, …, NY: Pearson, 2002.

[9]